UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Tadej Vrtačič

Mobilna podpora s storitvami v oblaku za taksiste

DIPLOMSKO DELO

VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM PRVE STOPNJE RAČUNALNIŠTVO IN INFORMATIKA

Mentor: doc. dr. Dejan Lavbič

Ljubljana, 2013

Univerza *v Ljubljani*
Fakulteta *za računalništvo in informatiko*

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00396 / 2013
Datum: 5.4.2013

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **TADEJ VRTAČIČ**

Naslov: **MOBILNA PODPORA S STORITVAMI V OBLAKU ZA TAKSISTE**

CLOUD BASED MOBILE SUPPORT FOR TAXI DRIVERS

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

Uporaba storitev taksistov je v večini primerov omejena na telefonsko zahtevo po prevozu stranke in nato prevzem na dogovorjenem mestu. Podatki o trenutni lokaciji taksistov, ki so morebiti prosti in lahko na željeno lokacijo pridejo najhitreje, stranki niso na voljo. Z globalnim pogledom nad lokacijo in zasedenostjo taksista bi se lahko le ti tudi bolj optimalno porazdelili in na ta način izboljšali svoje delovanje. V okviru diplomske naloge naj študent implementira prototip mobilne aplikacije in zaledni dela sistema, ki omogoča globalen pregled nad stanjem taksistev, prikazan na zemljevidu in učinkovito komunikacijo med strankami ter taksisti. Zaledni del sistema naj bo na voljo na spletu v obliki storitve v oblaku.

Mentor:

doc. dr. Dejan Lavbič

Dekan:

prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

IZJAVA O AVTORSTVU DIPLOMSKEGA DELA

Spodaj podpisani **Tadej Vrtačič**,

z vpisno številko **63080290**,

sem avtor diplomskega dela z naslovom:

Mobilna podpora s storitvami v oblaku za taksiste.

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno, pod mentorstvom doc. dr. Dejana Lavbiča,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki »Dela FRI«.

V Ljubljani, dne 15. oktobra 2013 Podpis avtorja:

Zahvala

Zahvaljujem se mentorju doc. dr. Dejanu Lavbiču za pomoč in usmerjanje pri opravljanju diplomskega dela. Posebna zahvala gre mojim staršem, ki so mi omogočili študij in mi skozi vsa leta stali ob strani ter tudi vsem mojim prijateljem, ki so me spodbujali in motivirali pri študiju.

KAZALO

Seznam uporabljenih kratic in simbolov

GPS (angl. Global Positioning System) - globalni sistem za pozicioniranje.

WiFi - zaščitni znak s katerim so označeni certificirani proizvodi za brezžično računalniško mrežo.

GSM (angl. Global System for Mobile) - svetovni standard mobilnih komunikacij.

EDGE (angl. Enhanced Data rates for Global Evolution) - tretja generacija mobilne tehnologije, ki omogoča hiter prenos podatkov.

3G - tretja generacija mobilne tehnologije, ki podpirajo storitve katere zagotavljajo prenos podatkov najmanj 200kbit/s.

OS - operacijski sistem.

Xml (angl. Extensible Markup Language) - razširljiv označevalni jezik.

Linux - prosto dostopen operacijski sistem.

API (angl. Application programming inteface) - vmesnik uporabniškega programa.

HTML (angl. HyperText Markup Language) - označevalni jezik za izdelavo spletnih strani.

CSS (angl. Cascading Style Sheets) - slogovni jezik, ki skrbi za prezentacijo spletnih strani.

JavaScript - objektni skriptni programski jezik.

PHP (angl. Hypertext Preprocessor) - razširjen odprtokodni programski jezik.

URI (angl. Uniform resource identifier) - niz znakov, ki se uporabljajo za identifikacijo spletnega vira.

SQL (angl. Structured Query Language) - strukturirani povpraševalni jezik za delo s podatkovnimi bazami.

SUPB - sistem za upravljanje s podatkovnimi bazami.

MySQL - odprtokodna implementacija relacijske podatkovne baze.

PB - podatkovna baza.

HTTP (angl. HiperText Transfer Protocol) - protokol za prenos informacij po spletu.

SDK (angl. Software Development Kit) - komplet orodij za razvoj programske opreme.

JDK (angl. Java Development Kit) - komplet orodij za razvoj aplikacij Java.

AVD (angl. Android Virtual Device) - virtualna naprava Android.

JSON (angl. JavaScript Object Notation) - preprost format za izmenjavo podatkov.

UI (angl. User Interface) - uporabniški vmesnik.

Povzetek

Diplomsko delo zajema celoten potek razvoja prototipa mobilne aplikacije, namenjene taksi službi ter njihovim strankam. Namen je bil, zagotoviti izmenjevanje ključnih informacij storitve prevoza, med uporabniki preko mobilne tehnologije.

V začetnem delu diplomske naloge je opisana ideja prototipa, pristop k razvoju mobilne aplikacije na platformi Android, predstavitev brezplačnih storitev Google ter drugih tehnologij, ki so uporabljene v aplikaciji. Predstavljen je pristop avtorizacije z OAuth 2.0 in dostop do uporabljenih storitev Google.

V osrednjem delu diplomska naloga zajema uporabo vseh razvojnih orodij, metodologijo dela ter predstavitev vseh aktivnosti prototipa mobilne aplikacije. Z upokojitvijo uporabljene storitve Google Latitude, v času izdelave diplomske naloge, se je poleg tega razvila tudi alternativna rešitev. Opredeljene so upokojene in nove delujoče rešitve prototipa. Predstavljen je tudi način komunikacije med uporabniki aplikacije, dostop do podatkovne baze MySQL ter spletni nadzor nad taksisti.

Del diplomskega dela so tudi težave pri razvoju, možne nadgradnje in izboljšave naše rešitve ter primerjava z obstoječimi rešitvami.

Ključne besede:

Android, OAuth, Google API, Google Latitude, Google Maps, GPS, MySQL, taksi

Abstract

The thesis includes the entire course of development of a prototype mobile applications, designed to taxi services and their customers. The purpose was to ensure exchanging key information of transport service between the users through mobile technology.

The beginning of the thesis describes the idea of the prototype, approach to the development mobile applications on the Android platform, presentation of free Google service and other technologies which are used in the application. Represents an approach to the OAuth 2.0 authorization and access to the Google services.

In the main part the thesis covers the use of development tools, methodology employed and the representation of all the activities of prototype mobile applications. With the retirement of using Google Latitude, between the time of producing thesis, the alternative solutions has been developed. The definitions of retired and new functioning prototype solutions are determined. Also manner of communication among the users of the application, access to the MySQL database as well as web control over taxi drivers are being introduced.

Part of the thesis also includes developmental difficulties, disturbances of upgrade and improvements to our solution as well as comparison with existing solutions.

Keywords:

Android, OAuth, Google API, Google Latitude, Google Maps, GPS, MySQL, taxi

1 UVOD

Napredek v mobilni tehnologiji je več kot očiten. Dandanes že skoraj ni telefona, ki nima večjedrnega procesorja, veliko pomnilnika in seveda svoj operacijski sistem. Pri mobilnih operacijskih sistemih v zadnjem času prevladuje operacijski sistem Android. Tudi način medsebojne komunikacije se je s tem bistveno spremenil. Danes je povezava pametnega telefona z internetom povsem nekaj samoumevnega, neodvisno od časa in lokacije.

Mobilni telefon je skozi ves ta dolgoletni razvoj postal tako zmogljiv, da je kos izvajati operacije, kot jih je včasih osebni računalnik. Dandanes imamo vrsto mobilnih aplikacij, ki nam lajšajo delo tako na poslovnem kot tudi na osebnem področju. Zaradi tega je naša ključna ideja bila izdelati aplikacijo za OS Android, namenjena taksi službi in njihovim strankam. Kot vemo še dandanes taksisti uporabljajo staro tehnologijo komunikacije. Namreč s svojo centralno postajo, ki jim narekuje delo, komunicirajo s pomočjo radijskih zvez. To komunikacijo uporabljajo tudi policisti, reševalne službe in še mnogi drugi.

Že skoraj v vsaki dejavnosti obstajajo različne rešitve pametnih mobilnih aplikacij. Na slovenskem trgu je na voljo malo mobilnih aplikacij, namenjenih taksistom in njihovim strankam. Cilj naše rešitve je razvoj pametne aplikacije, ki bi delovala na mobilnih telefonih oz. vseh napravah z OS Android. Mobilna aplikacija bi taksistom lajšala delo, predvsem zaradi komunikacije z njihovimi strankami, boljšim nadzorom ter večjim zanimanjem strank. Zagotavljanje zadovoljstva strank je za uspešnost taksi podjetja velikega pomena. Predolgo čakanje taksista, neposredna komunikacija s centralno postajo in še mnoge druge stvari stranko utrujajo in povzročajo nezadovoljstvo. Rešitev mobilne aplikacije nam zagotavljala hitro in preprosto naročilo storitve prevoza. Bistvo mobilne aplikacije je zagotovitev prevoza v danem trenutku glede na lokacijo stranke. Taksisti bi s tem imeli posledično več dela. Njihovo iskanje strank in komunikacija s centralo bi zamenjala mobilna aplikacija. Tudi uporaba raznovrstnih navigacijskih sistemov bi bila na strani taksistov sedaj odveč. Mobilna aplikacija vsebuje algoritem iskanja poti do stranke ali njenega cilja. Prvotna ideja je bila medsebojno komuniciranje (taksistov in strank) z uporabo lastnih strežniških storitev. Taksisti bi svoje trenutne lokacije shranjevali v podatkovno bazo, s pomočjo GPS in omrežja. Njihove vire lokacij pa prikazujemo na zemljevidu Google Maps. Boljšo motivacijo za razvoj nam je v začetku ponujala storitev Google Latitude. Njene funkcionalnosti so zadostovale in nam olajšale razvoj. Omogočala nam je prikazovanje virov trenutnih lokacij taksistov ter njihovo zgodovino lokacij. Uporaba svojih strežniških storitev (kreiranje podatkovne baze, računi uporabnikov, fizični podatkovni prostor) smo nadomestili s storitvijo Google Latitude. Za njeno uporabo smo potrebovali račun Google, katerega si lahko enostavno ustvarimo in je popularen saj ponuja uporabo še veliko drugih uporabnih storitev Google. Do uporabniških vmesnikov storitev Google smo z mobilnim telefonom dostopali s popularnim protokolom

OAuth 2.0. Samo komunikacijo med izbranim taksistom in stranko pa smo omogočili s storitvijo za medsebojno komuniciranje, poznano kot Google Talk. Torej, za uporabo mobilne aplikacije potrebujemo preprosto le račun Google.

Pri testiranju že delujočega prototipa se nam je ob nepazljivosti pojavila velika težava. Ključno storitev Google Latitude je podjetje Google upokojilo, kar nam je prineslo nedelujoči prototip. K problemu smo pristopili s svojo prvotno idejo, uporabe lastnih strežniških storitev. Realizirali smo podatkovno bazo MySQL, iz katere navadni uporabniki (stranke) pridobijo vse informacije o taksistih. Taksisti s pametno aplikacijo shranjujejo svoje trenutne lokacije v bazo podatkov in imajo vpogled v zgodovino lokacij. Za pravilno in fleksibilno delovanje mobilne aplikacije smo vseskozi testirali z realnimi podatki in mobilnimi telefoni z OS Android. Osredotočeni nismo bili zgolj na razvoj in testiranje mobilne aplikacije, ampak tudi na nadzor nad taksisti, ki uporabljajo mobilno aplikacijo. Nadzor taksistov poteka preko spletne aplikacije in nam predstavlja ideje o izboljšavah ter nadgradnje naše mobilne aplikacije.

V drugem poglavju so opisane uporabljene tehnologije pri razvoju prototipa. Tretje poglavje je namenjeno prikazu implementaciji in razvoju prototipa. Glavni del diplomske naloge je zajet v četrtem poglavju. Opisuje in predstavlja vse ključne razrede in funkcionalnosti prototipa mobilne aplikacije, njene izboljšave in testiranje. Zajema celoten postopek razvoja prototipa, od samega začetka do delujočega prototipa. Postopek poleg naše končne rešitve prototipa predstavlja in opisuje storitve, ki so bile zaradi težav opuščene. Na koncu v petem in šestem poglavju sledi predstavitev že obstoječih rešitev, zaključek in smernice za nadaljnji razvoj.

2 UPORABLJENE TEHNOLOGIJE

Z razvojem tehnologije so mobilni telefoni postali veliko več kot samo naprave za telefoniranje. Za to je zaslužen tudi sistem Android, ki je odprto kodna programska oprema, ki vključuje operacijski sitem, sistemski srednji sloj in mobilne aplikacije. Večina ljudi zmotno misli, da je sistem Android namenjen zgolj zahtevnejšim uporabnikom, ki imajo veliko predhodnega znanja in izkušenj z napravami kot so namizni in prenosni računalniki. A to seveda ne drži, saj je Android namenjen različnim uporabnikom ne glede na spol, starost in predznanje. Je enostaven in prilagodljiv, a hkrati kompleksen in zmogljiv.

Ker je Android odprtokoden in brezplačen, omogoča tudi cenejše in lažje razvijanje samosvojih rešitev. Spodaj so opisane vse tehnologije, ki smo jih uporabljali pri razvoju našega prototipa. S tem hočemo zagotoviti enostavno in prilagodljivo, a hkrati kompleksno in zmogljivo aplikacijo.

2.1 Pristopi avtorizacije (odprti protokoli)

V zadnjem času je internet preplavilo ogromno spletnih aplikacij ter družbenih omrežij, ki vseskozi zahtevajo našo prijavo. Za delitev informacij aplikacij z raznimi storitvami, ki tečejo na strežniku, potrebujemo avtentikacijo. Da bi si izognili delitvi svojih uporabniških imen in gesel z storitvami na internetu, uporabljamo odprte protokole.

Protokoli, ki jih lahko razvijalci uporabljamo pri razvijanju aplikacij, ki na danem omrežju pridobivajo in oddajajo podatke uporabnikom, katerim so to predhodno dovolili. Poznamo različne pristope obvladovanja avtorizacije pri spletnih in namiznih aplikacijah, mobilnih telefonih ter ostalih napravah. Najbolj razširjen odprt protokol za avtorizacijo je OAuth [8]. Google APIs za avtentikacijo in avtorizacijo uporabljajo protokol OAuth 2.0 [11].

2.1.1 Protokol OAuth

Oauth je odprt standard za avtorizacijo. Omogoča metodo dostopa odjemalcev do virov na strežniku v lastnikovem imenu (kot drug odjemalec ali kot končni uporabnik). Omogoča tudi postopek avtorizacije do strežnika za končne uporabnike nekih tretjih oseb. Podatki o uporabniku (uporabniško ime, geslo) so skriti in se ne delijo. OAuth je protokol ki služi varnemu in zasebnemu dostopu preko večjih omrežij, in se vseskozi dopolnjuje. V tem se razlikuje od OpenID. OpenID je odprto, decentralizirano, brezplačno ogrodje namenjeno digitalnim identitetam. OAuth se razlikuje tudi od OATH, ki je referenčna arhitektura za avtentikacijo. Da razčistimo pojme avtentikacija in avtorizacija. Avtentikacija je v računalništvu proces, v katerem se mora strežnik prepričati, da je uporabnik res tisti, za

katerega se izdaja. Primer avtentikacije je vpis oziroma prijava v nek sistem z uporabniškim imenom in geslom. Avtorizacija pa se uporablja za izvajanja overjenih (avtenticiranih) pravic uporabnikom, ki so jim bile pravice odobrene.

Začelo se je z novembrom, leta 2006, ko je Blaine Cook implementiral razvoj Twitter OpenID [10]. Skupina razprave OAuth je bila ustanovljena 2007, ko je skupina izvajalcev napisala osnutek predloge odprtega protokola. V juliju, leta 2007, je skupina pripravila začetno specifikacijo, 3. oktobra istega leta, pa je bil ustvarjen končni osnutek OAuth 1.0. Protokol OAuth 1.0 je bil objavljen, aprila 2010, kot RFC 5849. OAuth 2.0 okvir pa je bil objavljen v oktobru, leta 2012, kot RFC 6749 in RFC 7650. Dodatne RFC-je še vedno delajo.

2.1.2 OAuth 2.0

OAuth 2.0 ja naslednja evolucija protokola OAuth, le-ta je relativno enostaven protokol, s katerim se lahko razvijalec brez napora poveže s končnimi točkami Google OAuth 2.0. Protokol OAuth 1.0 se je izkazal za omejenega, zato so naredili izboljšave na področju:

- avtentikacija in podpisi,
- uporabniška izkušnja in izdaja alternativnih žetonov,
- skalabilnost in zmogljivost.

Novosti OAuth 2.0:

- Tok uporabniškega vmesnika – za odjemalce spletnega brskalnika.
- Tok spletnega strežnika – za odjemalce, ki so del strežniške aplikacije, dostopne preko HTTP zahtev.
- Tok uporabniškega imena in gesla – uporablja se v primeru ko uporabnik zaupa odjemalcu.
- Tok odjemalčevih poverilnic – uporabnik uporabi svoje poverilnice za pridobitev žetona.
- Tok naprave – primeren za odjemalce, ki tečejo na omejenih napravah.
- Tok uveljavljanja – odjemalec predstavi strežniku uveljavitev v zameno za žeton.

2.1.3 Dostop do Google API z uporabo OAuth 2.0

Sestavljen je iz štirih osnovnih korakov:

1. korak: Registracija aplikacije.

Vsi projekti, ki dostopajo do Google API, morajo biti registrirani preko konzole Google APIs. Rezultat tega procesa registracije so vrednosti parametrov, ki so znani tako pri Google kot pri našemu projektu. Ti parametri so: *client_id, client_secret, redirect_uri*.

2. korak: Pridobivanje žetona za dostop do avtorizacijskega strežnika Google.

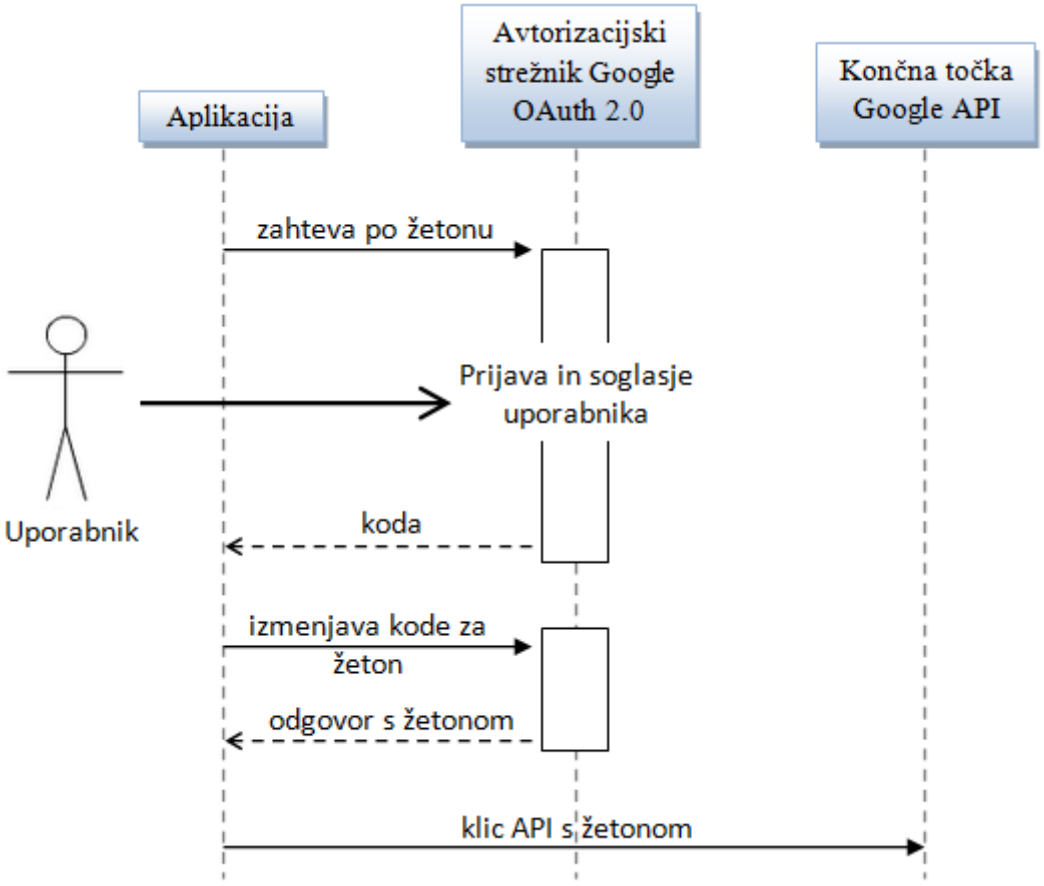
Predno lahko aplikacija dostopa do Google API, mora pridobiti žeton, ki omogoča dostop do želenega API-ja. Žetonu dostopa preko spremenljivega parametra imenovanega »scope«, podamo nabor virov in operacij, ki jih bomo nadalje uporabljali. V zahtevi se lahko določi več področij (»scope«). Zahtevo mora uporabnik prijaviti v Google, nato sledi privolitev uporabnika na odobritev vseh zahtev. V primeru potrditve, bo vaša aplikacija poslala potrditveni žeton ali avtorizacijsko kodo (uporablja se za pridobitev žetona).

3. korak: Pošiljanje žetona za dostop do API.

Po pridobitvi žetona za dostop lahko aplikacija v HTTP zahtevi pošlje žeton do Google API. Žetoni veljajo samo za nabor dejavnosti oz. operacij opisanih v zahtevi. Na primer, če se izda žeton za dostop do Google+, ne bo dovolil dostopa do Google Latitude.

4. korak: Osveževanje žetona za dostop (neobvezno).

Žetoni imajo omejeno življenjsko dobo. V nekaterih primerih aplikacija potrebuje dostop do API še po koncu trajanja enega žetona. Zato potrebujemo osveževanje žetonov, s katerim pridobimo nove žetone za dostop.



Slika 01: Prikaz dostopa uporabnika do Google API.

2.1.4 OAuth 2.0 in Android

OAuth 2.0 podpira različne tipe aplikacij. To so aplikacije spletnega strežnika, aplikacije s strani odjemalcev, nameščene aplikacije (Android, Windows) in aplikacije, ki tečejo na napravah (npr. igralne konzole, video kamere, tiskalniki).

Kot že zgoraj navedeno Google OAuth 2.0 strežnik za avtorizacijo podpira namizne in mobilne aplikacije (Android, Windows, Mas OS, iOS, Blackberry). Da gre za nameščeno aplikacijo je potrebno podati že pri registraciji projekta v konzoli Google APIs. Rezultat registracije se razlikuje v različnih vrednostih parametra za preusmeritev (*redirect_uri*). Parametra *client_id* in *client_secret* sta vgrajena v našo izvorno kodo aplikacije. Avtorizacijski strežnik Google podpira parametre, ki so predstavljeni v spodnji tabeli.

Parameter	Vrednosti	Opis
response_type	koda	Določa ali končni rezultat Google OAuth 2.0 vrne avtorizacijsko kodo. Vrednost kode uporabljajo nameščene aplikacije.
client_id	*client_id* pridobimo iz konzole APIs	Označuje odjemalca, ki je poslal zahtevo. Vrednost se mora natančno ujemati z vrednostjo v konzoli.
redirect_uri	Ena izmed vrednosti registrirana v konzoli.	Določa kam je poslan odgovor. V konzoli lahko zbiramo med *urn:ietf:wg:oauth:2.0:oob* ali *http://localhost* vratom (port).
scope	Omejen prostor za nabor dovoljenj aplikacijskih zahtev.	Označuje dostop do Google API vaše zahtevane vloge. Uporabniku je prikazana stran s soglasjem vrednosti opravil v tem parametru.

Tabela 1: Parametri, ki jih uporabnik potrebuje za avtorizacijo.

Ko Google obravnava uporabnikovo avtentikacijo, izbiro seje in njegovo soglasje, poda v rezultat odobritveno kodo. Naša aplikacija pa lahko izbira, kako bomo pridobili vrnjeno kodo. Ali preko spletne strani ali na HTTP vratih (angl. port) *http:/localhost*. Ko aplikacija pridobi avtorizacijsko kodo (koda odobritve), si jo lahko izmenjuje za žetone dostopa in osveževanja. Ob prejetju žetona za dostop, si tega lahko shranimo za nadaljnjo uporabo. Seveda dokler jim življenjska doba ne poteče. Po preteku pridobimo novega z osveževanjem.

Parameter *redirect_uri* vsebuje vrednost, s katero v naši aplikaciji določamo, kako nam bo koda oz. oznaka dovoljenja (angl. authorization code) vrnjena oz. prikazana. Vrednost tega parametra je lahko:

- **http://localhost**

Ta vrednost sporoča avtorizacijskemu strežniku Google, da mora biti oznaka dovoljenja vrnjena, kot parameter poizvedbe na odjemalčevem spletnem strežniku. Tega ne podpirajo vse platforme, če pa že, pa je to priporočen mehanizem za pridobitev oznake dovoljenja.

- **urn:ietf:wg:oauth:2.0:oob**

Oznaka dovoljenja pri tej vrednosti parametra *redirect_uri*, je vrnjena v naslovni vrstici brskalnika. To je uporabno, ko odjemalec ne more poslušati na HTTP vrata (angl. HTTP port), brez večjih nastavitev odjemalca. To lastnost imajo Windows aplikacije.

Ko aplikacija dobi kodo oziroma oznako dovoljenja, lahko to oznako izmenjamo za žetona za dostop in osveževanje. Zahteva po žetonu je HTTP sporočilo, ki vsebuje naslednje parametre: *code, client_id, client_secret, redirect_uri, grant_type.*

Z uspešnim odzivom na zahtevo pridobimo polja, ki jih za delo z želenimi storitvami potrebujemo. Spodaj je tabela pridobljenih vrednosti z zahtevo.

Polje	**Opis**
access_token	Parameter oziroma žeton, ki se ga lahko pošlje k Google API.
refresh_token	Žeton ki se uporablja za pridobitev novega žetona za dostop (access_token). Ti žetoni so veljavni dokler uporabnik ne prekliče dostopa.
expires_in	Preostanek življenjske dobe na dostopni žeton.
token_type	Označuje tip vrnjenega žetona.

Tabela 2: Pridobljene vrednosti po uspešni avtorizaciji.

Uspešen odziv na zahtevo se vrne kot JSON niz. **JSON** (angl. Java Script Object Notation) je besedilo, ki temelji na odprtem standardu izmenjavi berljivih podatkov. Izhaja iz skriptnega jezika JavaScript, ki predstavlja preproste podatkovne strukture in asociativne nize. Primer odziva strežnika na zahtevo z besedilom JSON:

```
{
  "access_token":"1/fFAGRNJru1FTz70BzhT3Zg",
  "expires_in":3920,
  "token_type":"Bearer",
  "id_token":"iW3cxlIoAW3cxlIgfrdg4e"
  "refresh_token":"1/xEoDL4iW3cxlI7yDbSRFkVKM2C-259HOF2aQbI"
}
```

Ko imamo pridobljen žeton za dostop, lahko dostopamo do Google API. Ker ima vsak žeton neko življenjsko dobo, lahko novega preprosto pridobimo.

2.1.5 Google OAuth API v2

Za avtentikacijo in avtorizavijo naše mobilne aplikacije do Google APIs uporabljamo uporabniški vmesnik *Google OAuth API v2*. Z njim v aplikaciji dostopamo do storitev in svojih podatkov o računu Google. Spodaj nam primer nazorno prikazuje HTTP zahtevo, s pridobljenim žetonom za dostop.

https://www.googleapis.com/oauth2/v1/userinfo?access_token=ya29.AHES6ZQU6yh1p-Kf25_LKW23pwouxHNm4HKOrHvtODBh6ug

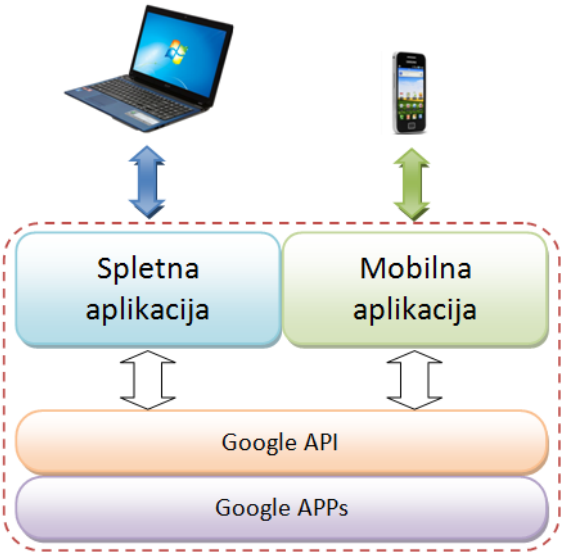
Vrnjeni podatki v formatu JSON zgornje HTTP zahteve:

```
{
 "id": "105081996470397441025",
 "email": "diplomatadej@gmail.com",
 "verified_email": true,
 "name": "diploma Tadej",
 "given_name": "diploma",
 "family_name": "Tadej",
 "link": "https://plus.google.com/105081996470397441025",
 "gender": "male",
 "birthday": "0000-04-01",
 "locale": "sl"
}
```

2.2 Google API

Uspeh podjetja Google temelji v celoti na njegovih brezplačnih spletnih in mobilnih storitvah. Uporabnikom svetovnega spleta in tudi uporabnikom pametnih telefonov platforme Android je namreč na voljo bogata paleta brezplačnih storitev, kot so na primer Gmail[1], YouTube[2], Google Docs[3], Earth[4], Google Maps[5], Google+[6], Google Latitude[7], Google Drive[8] in druge. Storitve niso brezplačne le uporabnikom temveč tudi njihovim programerjem, seveda z omejitvami določenih storitev, ki so jih pri podjetju Google uvedli. Na voljo je množica programskih vmesnikov (Google API), ki omogočajo programerjem ustvarjanje aplikacij v interakciji s storitvami podjetja Google. API je specifikacija, ki jo uporablja programska komponenta za medsebojno komunikacijo.

[1] https://mail.google.com/, [2] http://www.youtube.com/, [3] https://docs.google.com/,
[4] http://earth.google.com/ , [5] maps.google.com, [6] plus.google.com,
[7] latitude.google.com/latitude/ , [8] drive.google.com



Slika 02: Mobilna aplikacija kot vzporedni aplikacijski kanal.

Dandanes imajo že vsi pametni telefoni z OS Android s strani ponudnika naložene aplikacije Google. Spodaj so predstavljene vse aplikacije Google, ki jih uporabljamo v našem prototipu.

2.2.1 Google Maps

Google Maps je kartografski sistem, ki je za mnoge uporabnike osebnih računalnikov ter mobilnih naprav postal nepogrešljiv. Omogoča enostavno navigacijo, pregledovanje terena, preračunavanje razdalje, iskanje poti ko se izgubimo, in še mnogo več. Google Maps [23] je storitev za zemljevide, ki jih lahko uporabljamo na več načinov:

- s spletno aplikacijo na spletnem brskalniku,
- z aplikacijo Android ali iOS na mobilni napravi.

2.2.1.1 Google Maps Android API v2

Uporabniški vmesnik ponuja bogate in interaktivne funkcije zemljevida uporabnikom aplikacije Android. Ta različica omogoča mnogo izboljšav prejšnje verzije. API samodejno upravlja dostop do strežnikov Google Maps, prenos podatkov, prikaz zemljevidov in odzivnost na kretnje na zemljevidu. API klic lahko uporabimo tudi za dodajanje oznak, poligonov, prosojnice osnovne mape ter spreminjanje uporabnikovega pogleda na določenem območju zemljevida.

Omogoča dodajanje različnih grafov na zemljevid:

- Ikone, ki so prikazane na določeni poziciji na zemljevidu (angl. Markers).
- Risanje daljic (angl. Polylines).
- Zaprti segmenti (angl. Polygons).
- Rastrske grafike na določeni poziciji na zemljevidu (angl. Ground Overlays).
- Slike, ki so prikazane po zemljevidu (angl. Tile Overlays).

Za ustvarjanje aplikacije Android z zemljevidom Google Maps, je potrebno več korakov. Ker je *Google Maps Android API v2* del storitev *Google Play SDK*, je potrebno to namestiti in vstaviti v naš projekt.

Nato je potrebno pridobiti API ključ iz konzole Google APIs in ga dodati v projekt. Za ustvarjanje novega Android ključa v konzoli, potrebujemo prstni odtis SHA-1 ter ime paketa našega projekta.

```
Primer:
BB:0D:AC:74:D3:21:E1:43:67:71:9B:62:91:AF:A1:66:6E:44:5D:75;com.exam
ple.android.mapexample
```

Podpis certifikata prstnega odtisa SHA-1 uredimo preko terminala s ukazom *Keytool*.

```
keytool -exportcert -alias androiddebugkey -keystore pot-do-
produkcije-shrambe-ključev -list -v
```

Rezultat zgornjega ukaza je:

```
SHA1: BB:0D:AC:74:D3:21:E1:43:67:71:9B:62:91:AF:A1:66:6E:44:5D:75
```

Po pridobitvi API ključa v konzoli Google APIs, ga za pravilno delovanje aplikacije vstavimo v naš projekt. Dodamo ga v datoteko *AndroidManifest.xml,* s katero določamo glavne nastavitve aplikacije.

```
<meta-data
    android:name="com.google.android.maps.v2.API_KEY"
    android:value="API_KEY"/>
```

Če mislimo, da je naš Android ključ ogrožen, in v svojih poročilih konzole opazimo sumljive aktivnosti, potem ga lahko brez težav zamenjamo. Na novo generiramo nov ključ Android.
V aplikacijo je potrebno dodati še spodaj prikazano kodo, v kateri opredelimo ime paketa v našem projektu, kjer se nahaja mapa z zemljevidom Google Maps.

```
<permission
        android:name="com.example.mapdemo.permission.MAPS_RECEIVE"
        android:protectionLevel="signature"/>
<uses-permission
android:name="com.example.mapdemo.permission.MAPS_RECEIVE"/>
```

Projekt aplikacije Android, ki uporablja uporabniški vmesnik *Google Maps Android API v2*, potrebuje v svoji datoteki *AndroidManifest.xml* še naslednje nastavitve:

- Dovoljenja, ki dajejo projektu oz. aplikaciji dostop do sistemskih funkcij Androida in strežnika Google Maps.
- Obvestilo, da aplikacija zahteva 2. različico OpenGL ES.
- API ključ zemljevida.

```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission
android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission
android:name="com.google.android.providers.gsf.permission.READ_GSERV
ICES"/>
```

Spodnji dve dovoljenji pri uporabi *Google Maps Adroid API v2* ni potrebno uporabiti. So pa priporočljivi.

```
<uses-permission
android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION"/>
```

Ker Google Maps Android API v2 zahteva OpelGL ES verzije 2, je potrebno dodati še:

```
<uses-feature
        android:glEsVersion="0x00020000"
        android:required="true"/>
```

Ta obvesti zunanje storitve na zahtevo. OpenGL ES (angl. OpenGL for Embedded Systems) je brezplačen in deluje na različnih platformah, za polno delovanje 2D in 3D grafike. Namenjen je za vgrajene sisteme kot so mobilni telefoni, dlančniki in igralne konzole.
Osrednja aktivnost naše aplikacije je aktivnost, ki za uporabniški vmesnik uporablja zemljevid Google Maps. Aktivnost modelira objekt zemljevida znotraj aplikacije. V uporabniškem vmesniku aplikacije, zemljevid zastopa objekt MapFragment ali MapView.

Aktivnost samodejno upravlja naslednje operacije:

- povezovanje s storitvijo Google Maps,
- nalaganje ploščice zemljevidov,
- prikaz ploščic na zaslonu naprave,
- prikaz različnih kontrol (kompas, povečava),
- odziv na kretnje kontrol (pomikanje, povečava karte).

Poleg vseh teh samodejnih operacij lahko nadziramo obnašanje zemljevidov z objekti APIja in metodami. *Google Maps Android API v2* ponuja štiri različne tipe kart:

- Normalni pogled (angl. Normal view), je tipičen zemljevid cest.
- Hibridni pogled (angl. Hybrid view) so satelitski fotografski podatki z dodanimi kartami cest.
- Satelitski pogled (angl. Satellite view) so satelitski posnetki. Cestne in ostale oznake niso vidne.

- Terenski pogled (angl. Terrain view) so topografski podatki. Zemljevid vključuje barve, oznake, plastnice in perspektivno senčenje. Vidne so tudi nekatere ceste in oznake.

Poleg vseh naštetih operacij omenjeni API omogoča tudi risanje po zemljevidu. Rišemo lahko označevalce (angl. Markers) lokacij, linije, poligone, kroge… S tem poskrbi API za dobro interakcijo z uporabnikom, kar je ključnega pomena.

2.2.2 Google Latitude

Z veliko popularnostjo uporabe družbenih omrežij se je močno povečala tudi uporaba lokacijsko odvisnih aplikacij. Te nam omogočajo še lažje komuniciranje s svojimi prijatelji, sodelavci.. Primerov uporabe je na tem področju veliko. Eno zelo razširjenih lokacijsko-odvisnih aplikacij je tudi storitev Google Latitude [25]. Uporablja se kot spletna aplikacija na osebnem računalniku ali na mobilnem telefonu (Android, iPhone, …).

Lokacijska storitev Google Latitude izkorišča GPS v pametnem telefonu, podatke iz socialnih omrežij… Storitev je narejena na enem osnovnem konceptu: uporabniška lokacija.

Lokacija je zapis, kjer se v danem trenutku nahajaš. Vedno vsebuje zemljepisno širino, dolžino in časovni žig. Lahko vsebuje tudi več podrobnosti o stališču uporabnika, kot so nadmorska višina, hitrost ali metapodatki o natančnosti položaja.

Torej je njena primarna funkcionalnost omogočanje deljenja trenutne lokacije z ostalimi (prijatelji, sodelavci itd.). Storitev je v večini primerov integrirana v aplikacije z zemljevidom Google Maps, saj je tako najbolj funkcionalna. Pri uporabi storitve lahko uporabnik omogoči deljenje trenutne lokacije, ki jo pridobi s pomočjo senzorjev mobilne naprave, ali pa jo določi kar ročno. Če uporabnik storitve nima omogočene, njegova trenutna lokacija ne bo vidna drugim uporabnikov.

Glavne funkcionalnosti Google Latitude:

- posodabljanje in delitev lokacij z izbranimi prijatelji,
- prikaz zasebne zgodovine lokacij,
- prijava oz. sporočanje specifičnih lokacij (prenočišča, restavracije) prijateljem.

Storitev Google Latitude omogoča nastavitev zasebnosti lokacije, ali deljenje lokacije z drugimi. Našo lokacijo lahko različno nastavljamo:

- Zaznavanje naše lokacije - lokacija se samodejno posodablja.
- Nastavitev naše lokacije - ročna izbira lokacije na zemljevidu, ki jo delimo s prijatelji.
- Skrivanje naše lokacije oz. neposodabljanje naše lokacije - od trenutka te nastavitve, se naša lokacija ne bo posodabljala v storitev.

Vse omogočene posodobljene lokacije so javno dostopne in so vidne vsem, zato previdnost pri tem ni odveč.

Natančnost lokacije je odvisna od tega, kateri od virov podatkov o lokacijah je na voljo storitvi Google Latitude v vaši napravi. Natančnost izboljšamo z vklopljenim vmesnikom Wi-Fi, zlasti ko smo v zgradbi in je GPS signal šibek. Natančnost lokacije se z uporabo vmesnika Wi-Fi izboljša, kljub temu de niste povezani v nobeno omrežje Wi-Fi.

2.2.2.1 Viri podatkov

Google Latitude lahko uporabi naslednje vire podatkov o lokaciji:

- **GPS:** GPS lahko kaže lokacijo na več metrov natančno, odvisno od signala GPS in povezave.
- **Wi-Fi:** Natančnost omrežja Wi-Fi naj bi bila približno enaka ravni dometa običajnega usmerjevalnika omrežja Wi-Fi oz. okrog 200 metrov ali več.
- **ID celice:** Storitev uporablja Google-vo zbirko podatkov o lokacijah ID-jev celic brez omrežja Wi-Fi. Natančnost je odvisna od gostote baznih postaj in razpoložljivih podatkov.

2.2.2.2 Googlova značka z javno lokacijo

V storitvi Google Latitude lahko našo lokacijo objavimo na spletnem mestu. Uporabimo lahko običajno kodo značke Google ali pa kot JSON vir lokacije. Če imamo v storitvi svojo lokacijo omogočeno kot vidno, lahko to vidijo tudi tiste osebe, pred katerim smo se s storitvijo Latitude skrili.

Primer običajne Google značke lokacije:

```
<!-- Google Public Location Badge -->
<iframe
src="http://latitude.google.com/latitude/apps/badge/api?user=-
16990566678674667 73&type=iframe&maptype=roadmap&hl=sl" width="180"
height="300" frameborder="0"></iframe>
<!-- To disable location sharing, you *must* visit
https://www.google.com/latitude/apps/badge and disable the Google
Public Location badge. Removing this code snippet is not enough! -->
```

Primer JSON vira lokacije, ki se nahaja na naslovu *http://latitude.google.com/latitude/apps/badge/api?user=-16990566678674667 73&type=json*:

```
{
"type": "FeatureCollection",
"features": [
{
     "type": "Feature",
     "geometry": {"type": "Point", "coordinates": [15.1591167,
45.7946897]},
     "properties":
     {
          "id": "-16990566667867466773",
          "accuracyInMeters": 25,
          "timeStamp": 1373625666,
          "reverseGeocode": "8000 Novo mesto, Slovenija",
          "photoUrl":
"http://latitude.google.com/latitude/apps/badge/api?type=photo&photo
=Q46A0j8BAAA.XDrM1QjD6Xrv7dfeCuEAfg.FM8O_HDErY-VfmUTF8LlHg",
          "photoWidth": 96,
          "photoHeight": 96,
          "placardUrl":
"http://latitude.google.com/latitude/apps/badge/api?type=photo_placa
rd&photo=Q46A0j8BAAA.XDrM1QjD6Xrv7dfeCuEAfg.FM8O_HDErY-
VfmUTF8LlHg&&stale=true&lod=1&format=png",
          "placardWidth": 56,
          "placardHeight": 59
     }
}]
}
```

V primeru da to kodo objavimo na neki spletni strani, je to vidno vsem, ki lahko do nje dostopajo. Če ne želimo, da je naša lokacija dostopna »vsem«, je pri tem potrebno biti pazljiv.

2.2.2.3 Glavne operacije Google Latitude API

Z Google Latitude API lahko uporabljamo štiri različne metode za upravljanje njihovih virov.

Operacija	Opis	REST HTTP metode
list	Seznam vseh lokacij v zgodovini lokacij.	**GET** na URI zgodovine lokacij.
insert	Vstavimo novo lokacijo. Če lokacija s tem časovnim žigov že obstaja, se njegova vsebina posodobi iz pridobljenih podatkov.	**POST** na URI zgodovine oz. trenutne lokacije, kjer prenesemo k podatkom nov vir lokacije.
get	Pridobimo določeno lokacijo.	**GET** na URI vira lokacije.
delete	Zbrišemo določeno lokacijo.	**DELETE** na URI vira lokacije.

Tabela 3: Osnovne operacije Google Latitude API.

Spodnja tabela prikazuje operacije, ki so podprte za različne tipe virov.

Tipi virov	Podprte operacije			
	list	**insert**	**get**	**delete**
Vir lokacije (angl. Location Resource)		DA	DA	DA
Zbirka trenutnih lokacij (angl. Current Location Collection)		DA	DA	DA
Zbirka zgodovine lokacij (angl. Location History Collection)	DA	DA	DA	DA

Tabela 4: Podprte operacije s strani različnih tipov virov.

Vse operacije zahtevajo avtentikacijo, ki je opravimo s protokolom OAuth 2.0.

2.2.3 Google Drive

Računalništvo v oblaku je dandanes zelo priljubljena tema in ima zadovoljive lastnosti. Prilagodljivost, dosegljivost in razširjenost so tri izmed njih, ki jih je podedovala oblačna shramba. Shramba v oblaku se je izkazala za odlično rešitev, poleg prostora pa smo z njo dobili nenehno dosegljivost. Dropbox[9] je ena izmed prvih storitev, ki je pokazala kako priročen je lahko Oblak. Tudi pri Googlu so v ta namen naredili storitev Google Drive [20]. Storitev omogoča uporabniku shranjevanje, izmenjavo datotek in sodelovanje pri urejanju. Storitev lahko uporabljamo na različnih operacijskih sistemih. Tudi na naših pametnih mobilnih telefonih.

V naši aplikaciji smo storitev integrirali s pomočjo orodja Google Drive SDK. Ta nam ponuja skupino programskih vmesnikov (APIs), knjižnice odjemalcev, primere in dokumentacijo. Kasneje smo storitev v aplikaciji opustili, zaradi svoje alternative. Ta je bila nujno potrebna, ker se je storitev Google Latitude upokojila. Vpeljava storitve Google Drive je bila funkcionalna in smiselna z souporabo Google Latitude.

[9] http://www.dropbox.com/

2.2.3.1 Glavne operacije Google Drive API v2

Storitev omogoča veliko funkcionalnosti. V spodnji tabeli so prikazane metode za upravljanje z datotekami.

Metoda	Zahteva HTTP	Opis
get	GET /files/*fileId*	Pridobimo metapodatke datoteke z njenim ID-jem.
insert	POST https://www.googleapis.com/upload/drive/v2/files	Vstavljanje nove datoteke.
patch	PATCH /files/*fileId*	Posodobitev metapodatkov datoteke.
update	PUT https://www.googleapis.com/upload/drive/v2/files/*fileId*	Posodobitev metapodatkov ali vsebine datoteke.
copy	POST /files/*fileId*/copy	Ustvarjanje kopije določene datoteke.
delete	DELETE /files/*fileId*	Trajno brisanje datoteke z njenim ID-jem.
list	GET /files	Seznam datotek uporabnika.

Tabela 5: Glavne metode Google Drive API v2.

V naši aplikaciji smo osredotočeni zgolj na branje datoteke formata JSON, ki jo hranimo v Google Drive. Datoteka je vnaprej shranjena kot javno dostopna.

2.2.3.2 Vpeljava Google Drive API v aplikacijo Android

Google Drive API, za avtentikacijo uporabnika z računom Google, uporablja protokol OAuth 2.0.
Koraki vpeljave storitve Google Drive API v našo aplikacijo:

1. **korak: Aktiviranje Google Drive API**

Kot prvo je potrebno za vpeljavo storitve v naš projekt, omogočiti storitev v konzoli Google APIs.

2. **korak: Namestitev knjižnice Google Client**

Knjižnice pridobimo iz Google-ove spletne strani za razvijalce. Najprej si prenesemo datoteko zip, ki jo razpakiramo. Knjižnice JARs prekopiramo v mapo libs našega projekta. Za uporabo storitve Google Drive API dodamo v projekt knjižnico *google-api-services-drive-v2-[verzija].jar*.

3. **korak: Vzpostavitev storitve v aplikaciji**

Ko je storitev aktivirana v konzoli Google APIs, in ko so v našem projektu prisotne vse knjižnice, potem lahko začnemo s programiranjem naše aplikacije. Knjižice pridobimo na uradni spletni strani za razvijalce: https://developers.google.com/api-client-library/java/apis/drive/v2. Pri nadaljnjem razvoju potrebujemo v našo izvorno kodo dodati parametra Client ID in Client Secret, ki sta potrebna pri dostopu do želenega API-ja.

Poleg vsega naštetega je potrebno poskrbeti, da imamo v datoteki *AndroidManifest.xml* dodana dovoljenja, ki omogočajo uporabo storitve Google Drive.

```
<uses-permission android:name="android.permission.GET_ACCOUNTS"/>
<uses-permission android:name="android.permission.INTERNET" />
```

2.3 MySQL in PHP

MySQL je odprtokodni sistem za upravljanje s podatkovnimi bazami. Zbirka podatkov v MySQL je strukturirana, za delo s podatki pa uporablja jezik SQL (angl. Structured Query Language). SQL je najbolj uporabljen standardiziran jezik, ki se uporablja za dostop do zbirk podatkov. Sistem je namenjen enostavnejši ali zahtevnejši količini informacij v omrežju podjetja. Deluje na principu odjemalec–strežnik. Strežnik MySQL služi kot sistem za dodajanje in obdelavo podatkov v zbirki podatkov. Strežnik podpira različne odjemalske programe in knjižnice, administrativna orodja in velik razpon aplikacijskih vmesnikov. Relacijska zbirka podatkov se shranjuje v ločenih tabelah, kar povečuje hitrost in fleksibilnost. Ker je programska oprema MySQL odprtokodna, omogoča, da ima vsak možnost dostopa do programske opreme ter si jo prilagodi po lastni meri oziroma potrebi.

PHP (angl. HyperText Preprocessor) je razširjen odprtokodni jezik. Uporablja se za strežniške uporabe oziroma razvoj dinamičnih spletnih strani. PHP primarno teče na spletnem strežniku, kjer jemlje PHP izvorno kodo za vhod in generira spletno stran kot izhod. Omogoča tudi možnost zaganjanja skript v ukaznem načinu.

PHP v kombinaciji z MySQL je zelo popularna brezplačna alternativa spletnim rešitvam.

3 IMPLEMENTACIJA IN RAZVOJ APLIKACIJE

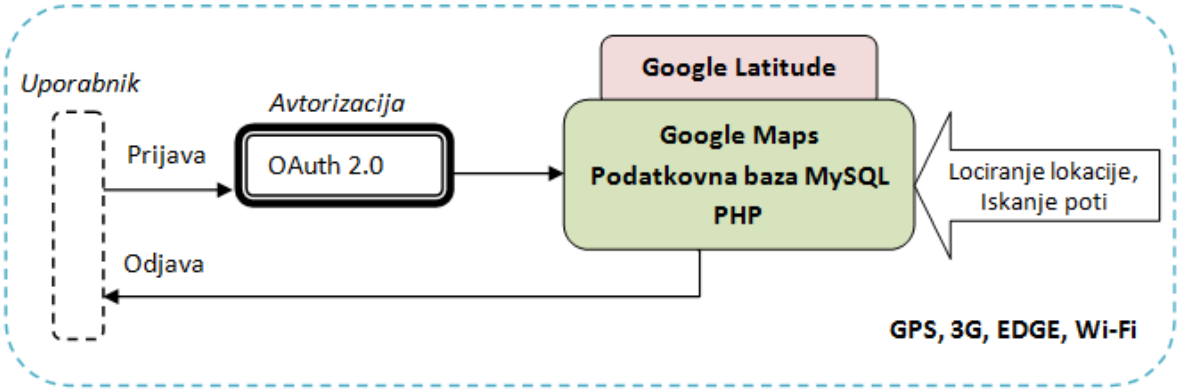
Mobilne aplikacije je potrebno obravnavati drugače kot ostale aplikacije, zaradi njihove namembnosti in omejitev mobilnih naprav. Mobilna aplikacija mora uporabniku prinesti neko novo vrednost glede na ostale zvrsti aplikacij. Gre za povsem nov model aplikacij. Mobilne aplikacije delimo na dva modela:

- *Klasičen model (angl. pull model):* Značilnost tega modela je, da uporabnik sam zažene, aktivira mobilno aplikacijo v izbranem trenutku.
- *Model kontekstno odvisnih mobilnih aplikacij (angl. push model):* Mobilna aplikacija se zažene, aktivira sama v odvisnosti od konteksta in pravila za proženje. Kontekstno odvisna je takrat, ko je odvisna od vsaj ene komponente konteksta. Najpomembnejša komponenta konteksta pa je informacijska potreba uporabnika.

3.1 Metodologija dela

Za dobro, funkcionalno in uporabniku prijazno aplikacijo se je potrebno dodobra organizirati in uporabiti dobro metodologija dela. Natančno je potrebno predelati vse njene funkcionalnosti in probleme oz. omejitve, do katerih lahko pride pri mobilni aplikaciji. Ustvariti si moramo nekako sliko, kako naj bi aplikacija v osnovi izgledala in katere bodo tiste ključne akcije, ki bodo nemoteno potekale.

Z razvojem Android aplikacij sem se osebno srečal v tretjem letniku študija pri predmetu elektronsko mobilno poslovanje. Tam sem pridobil osnovno znanje, kar mi omogoča hitrejši razvoj Android aplikacij. Skozi izdelavo diplomske naloge sem svoje znanje poglobil in ga nadgradil. Izkušenj nisem imel, zato sem si pomagal s primeri in literaturo z interneta. V začetku smo si zasnovali osnovni diagram poteka aktivnosti prototipa, le-tega smo sprotno nadgrajevali.



Slika 03: Osnovna ideja diagrama poteka aktivnosti prototipa.

Znane so bile osnovne zahteve aplikacije, zato smo se najprej osredotočili na razvoj glavnega uporabniškega vmesnika, ki je prikazoval zemljevid Google Maps. Lokacije smo na zemljevidu sprva prikazovali s pomočjo storitve Google Latitude. Po testiranju začetnega prototipa smo se lotili razvoja ostalih zahtev ter to sprotno testirali. Postopoma z dodajanjem novih funkcionalnosti in s popravkom starih prototipov, se je gradila končna aplikacija. Med testiranjem smo pri že delujočem prototipu naleteli na težavo. Storitev Google Latitude smo, zaradi upokojitve, bili primorani zamenjati s svojo alternativo. Ta alternativa vsebuje podatkovno bazo MySQL, ki se nahaja na strežniku »http://alfa.pip.si«.

Proces razvoja mobilne aplikacije poteka v več fazah, v vsaki pa potekajo različna opravila. Procesi razvoja mobilne aplikacije so naslednji:

- ANALIZA,
- NAČRTOVANJE,
- IMPLEMENTACIJA,
- TESTIRANJE,
- INŠTALACIJA,
- OBRATOVANJE IN VZDRŽEVANJE.

S temi koraki pridemo do celovite in učinkovite mobilne aplikacije.

3.2 Uporabljena orodja

Pri razvoju Android aplikacije je potrebno uporabljati različna orodja in tehnologije. Aplikacijo smo razvijali v razvojnem okolju Eclipse. Za začetek smo potrebovali Android SDK (angl. Software Development Kit), ki vsebuje vse potrebno za izdelavo kot tudi za testiranje aplikacij Android. Potrebno je bilo namestiti tudi razvojno ogrodje JDK (angl. Java SE Development Kit). Najnovejšo različico prenesemo iz uradne strani.

3.2.1 Razvojno okolje Eclipse

Za dober in hiter razvoj potrebujemo dobro razvojno okolje. Preden pa se posvetimo razvoju, je potrebno dodobra spoznati razvojno okolje. Eclipse IDE (angl. Integrated Development Environment) je zelo razširjeno razvojno okolje, ki podpira več programskih jezikov. Eclipse omogoča odlično podporo za programiranje Jave, poleg tega pa nudi veliko dodatkov za lažje delo za platformo Android. Njegova bistvena prednost je tudi to, da je prosto dostopen. Ključ do dobrega razvoja aplikacije je dobro pripravljeno delovno okolje, kar nam razvojno okolje Eclipse IDE vsekakor omogoča. Organiziranost ter dober pregled razredov projekta sodi med njegovo dobro plat.

3.2.2 Razvojna orodja in namestitev

Kot prvo je v razvojno okolje potrebno namestiti dodatek ali vtičnik ADT (angl. Android Development Tools). ADT je vmesnik za nameščanje drugih vtičnikov Android. Poleg namestitve vtičnikov Android je potrebno namestiti še gonilnik, ki omogoča razhroščevanje in enostavno poskusno poganjanje aplikacij na napravi.

Za razvoj aplikacij platforme Android so pri podjetju Google pripravili skupek knjižic in orodij (Android SDK), ki nam lajšajo delo. Android SDK (angl. Software Development Kit) je paket, ki združuje vsa orodja za razvoj mobilnih aplikacij. Vanj so vključene knjižnice, emulator, razhroščevalnik in dokumentacija.

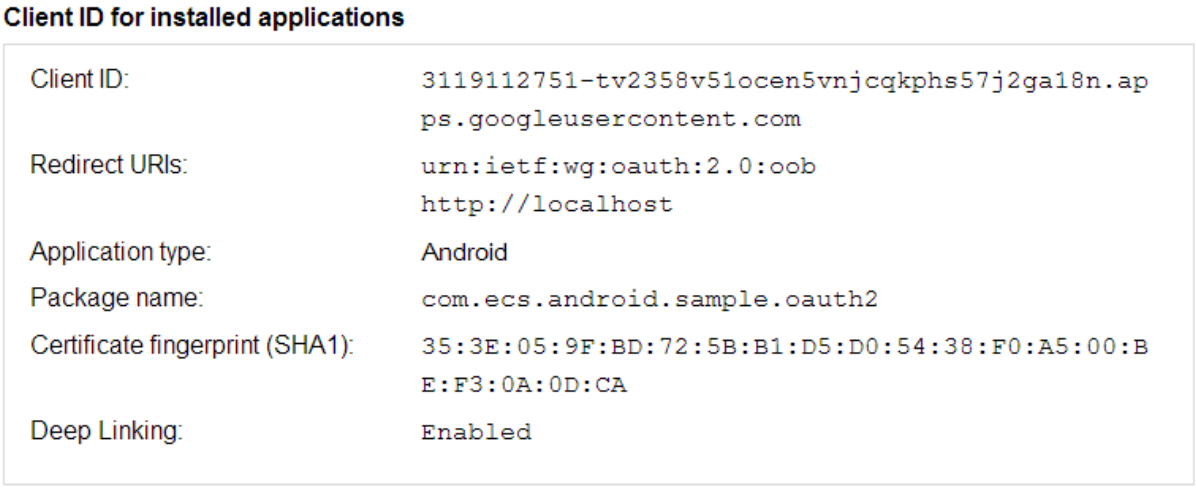
Aplikacije testiramo z emulatorjem oziroma vgrajeno virtualno napravo AVD (angl. Android Virtual Device). Emulator predstavlja neko kopijo mobilne naprave, kateremu nastavimo glavne lastnosti. To so verzija OS Android, velikost pomnilnika ter ločljivost zaslona. Ima večino funkcionalnosti in lastnosti, ki jih ima prava naprava Android. Ima pa tudi pomanjkljivosti. Določenih lastnosti virtualna naprava ne podpira. Zato smo testiranje izvajali na pametnem mobilnem telefonu Galaxy ACE GT-S5830 z OS Android različice 2.3.3. Njegove karakteristike so zadostovale za brezhibno testiranje aplikacije. 800 MHz procesor, 158MB integriranega prostora, 278MB pomnilnika (RAM) in 3.5" (inches) velik zaslon so dovolj za normalno delovanje aplikacije.

Za razvoj mobilne aplikacije smo uporabljali programski jezik Java. Potrebovali smo nameščeni Java JDK. JDK (angl. Java Development Kit) je program razvojnega okolja za pisanje aplikacij Java.

3.3 Uporaba Google API

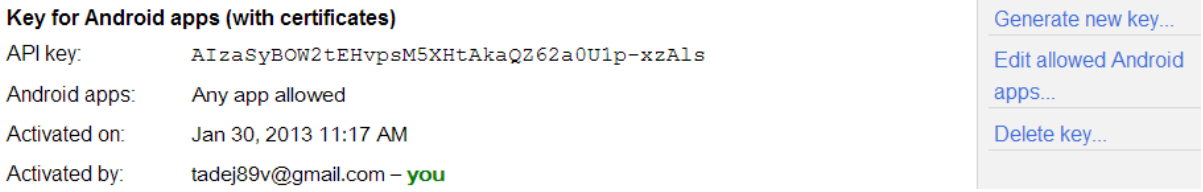
Uporaba aplikacijsko programskih vmesnikov (API) podjetja Google je dokaj enostavna. Kot prvo kar moramo imeti za uporabo le-teh je račun Google, s katerim dostopamo do vseh API-jev. Z računom Google se prijavimo tudi na konzolo namenjeno razvijalcem, ki uporabljajo Google API-je. V konzoli ustvarimo oz. registriramo projekt, s katerim bomo uporabljali želene storitve. Google razvijalcem nudi veliko, zato imamo na voljo več programskih vmesnikov (API). Vsakega posebej je potrebno, za pravilno delovanje storitve, v konzoli vključiti ali bolje rečeno omogočiti. Tukaj definiramo tudi vse ustrezne parametre, ki jih potrebujemo za pravilno delovanje posameznega API-ja. Storitve Google, s katerimi izmenjujemo informacije, potrebujejo avtentikacijo OAuth 2.0. V ta namen kreiramo odjemalčevo identifikacijsko številko (ID) protokola OAuth 2.0, ki jo uporabimo v našem projektu aplikacije Android. Pri kreiranju je potrebno paziti na pravilno izbiro vrste aplikacije. V našem primeru gre za nameščeno aplikacijo Android.

Na spodnji sliki so prikazani vsi podatki odjemalca, ki jih potrebujemo za razvoj s protokolom OAuth 2.0.



Slika 04: Podatki odjemalca oz. razvijalca projekta v konzoli Google APIs.

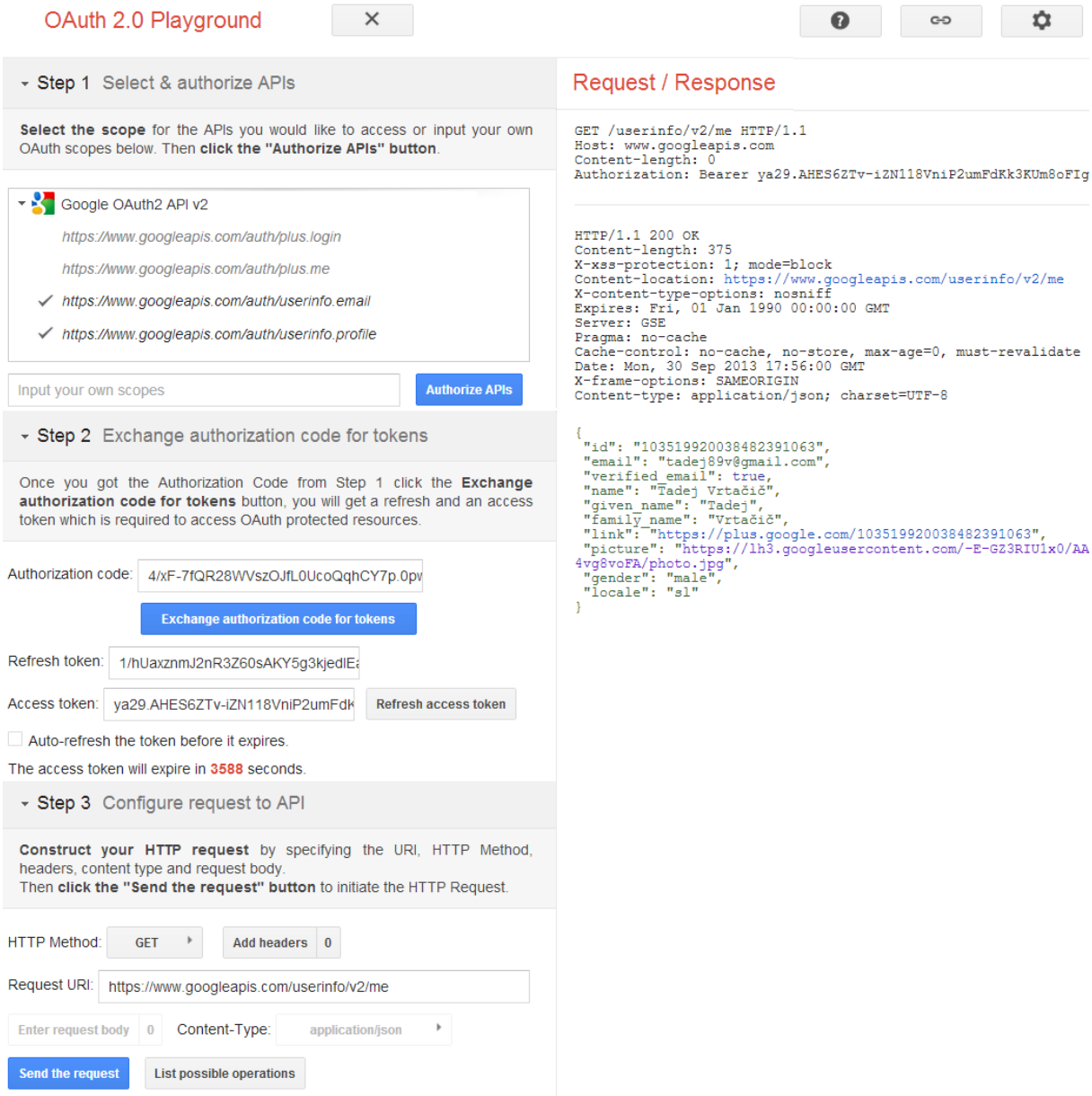
Za dostop do API potrebujemo še API ključ, ki je odvisen od določenega tipa aplikacije. Lahko izbiramo med spletno aplikacijo, strežniško aplikacijo, aplikacijo za iOS ali aplikacijo za Android.



Slika 05: Primer ključa API aplikacije Android v konzoli Google APIs.

Pravilno nastavljeni parametri, ki jih potrebujemo pri razvoju aplikacije z določenimi API-ji, še niso dovolj. Za vsako storitev imamo na voljo najnovejše knjižice. V knjižicah so vsi potrebni razredi in metode, ki nam omogočajo funkcionalnost želenega Google API. Projekt, ki vsebuje več API-jev in s tem tudi veliko knjižic, mora biti dodobra organiziran. V nasprotnem primeru kmalu pride do velike zmešnjave. Uporaba različnih knjižic, s podobnimi oz. tudi nekaterimi istimi razredi in metodami, lahko povzroča težave. Poskrbeti moramo, da imamo v projektu najnovejše različice knjižnic, in da so med seboj kompatibilne. Vrstni red knjižnic v projektu razvojnega okolja Eclipse je pomemben. V vsakem posameznem razredu se pri programiranju v razvojnem okolju Eclipse avtomatsko generirajo klici knjižnic. Kar pa ni nujno, da je tudi pravilno. Iz velikega nabora knjižnic našega projekta je lahko avtomatsko generiranje klicev napačno. Za pravilno delovanje programa je potrebno to skrbno pregledati. Veliki projekti potrebujejo dobro organizacijo. S tem se izogibamo napakam in delovanje programa poteka brezhibno.

Za dostop do Google APIs uporabljamo protokol OAuth 2.0. Pri programiranju z omenjenim protokolom si lahko pomagamo s spletno stranjo OAuth 2.0 Playground. Tukaj lahko natančno vidimo pregled in delovanje vseh metod Google API z uporabo omenjenega protokola. S tem omogoča boljše razumevanje uporabe OAuth 2.0 ter lažji razvoj našega projekta. Testiramo lahko posamezne URI zahteve določenih API metod. Spletna stran nam avtomatično ponuja listo vseh možnih zahtev, ki jih lahko izvedemo nad določenim API-jem. Ob pošiljanju zahtev določenemu API-ju nam spletna stran vrne rezultat.



Slika 06: Primer spletne strani https://developers.google.com/oauthplayground.

Podan rezultat je v preprostem formatu za izmenjavo podatkov imenovanemu JSON.

4 PROTOTIP APLIKACIJE

Programska koda celotnega projekta prototipa aplikacije se nahaja na spletnem naslovu: https://github.com/tadej89v/mobilna-podpora-taksistom.

4.1 Opis problema in osnovna ideja

Komunikacija je proces informacij od ene osebe k drugi. Komunikacijski proces je izmenjava sporočil skozi različne informacijske kanale od izvora do prejemnika sporočil. Komunikacija je zelo pomembna tako na poslovnem kot osebnem področju. Dandanes je vse več komunikacije s pametnimi mobilnimi telefoni, nam lajšajo vsakdanja opravila, tako si vsakodnevna dela brez njega že skoraj ne predstavljamo. Pri komunikaciji z mobilnimi telefoni oz. z mobilnimi aplikacijami gre za komunikacijo s podatki, ki se hranijo nekje v centrali (strežniki, varnost, hitrost, udobje).

Problem, ki smo ga opazili, je v komunikaciji med taksisti in njihovo centralo, ki jim narekuje delo. Ta problem je tudi pri policiji, logističnih organizacijah in vseh, ki še dandanes uporabljajo staro tehnologijo komuniciranja. To je komuniciranje preko radijskih zvez. Ker je razvoj mobilnih telefonov in s tem tudi mobilnih aplikacij močno napredoval, sem iznašel primerno rešitev za ta problem.

Osnovna ideja je, zagotoviti taksistom in njihovim koristnikom lažjo komunikacijo z mobilno aplikacijo Android. Na strani taksistov in njihovimi strankami bi komunikacijo olajšali z mobilno aplikacijo. Na strani centrale pa bi komunikacijo zamenjale brezplačne in brezhibne Google-ove storitve. To so Google Maps, Google Latitude, Google Drive, Google Talk. Pri razvoju pa se ni izšlo vse po načrtu. Ker so storitev Google Latitude upokojili, nam že delujoči prototip ni deloval. To smo nadomestili z vpeljavo svoje alternative. Ustvarili smo svojo podatkovno bazo MySQL, ki omogoča podobne funkcionalnosti, kot nam je to zagotavljala storitev Google Latitude. S tem je bila v projektu nadomeščena tudi uporaba Google Drive. Z njim smo namreč shranjevali seznam vseh ID-jev taksistov, ki so uporabljali Google Latitude. ID se je namreč razlikoval od osnovnega ID računa Google.

4.2 Programska logika

Da se podatki smiselno in v pravem vrstnem redu vrstijo na zaslon, je za to potrebna programska logika, ki poteka v ozadju. Programska logika je programirana v programskem jeziku Java. Na klasičnih računalnikih, različnih operacijskih sistemov, smo navajeni, da naenkrat teče več aplikacij, ki so hkrati vidne na zaslonu. Aplikacije android ne delujejo na enak način in se praviloma zaženejo kot aktivnosti. Aktivnosti se lahko prožijo iz različnih

komponent aplikacije. Uporabniku zagotavljajo uporabniški vmesnik, s katerim lahko upravljajo aplikacijo. V določenih primerih lahko aktivnost vrača tudi neko vrednost. Uporabnik lahko poljubno prehaja med aktivnostmi. Vse aktivnosti so shranjene v aplikacijski sklad, ki ga upravlja upravljavec aktivnosti (angl. Activity Manager). S pomočjo sklada lahko pridemo do predhodnega pogleda oziroma aktivnosti.

Aktivnosti imajo svojo življenjsko dobo. V svojem življenjskem ciklu ima eno izmed več možnih stanj (nazorno prikazano na spodnji sliki). Vsaka aktivnost ima štiri glavna stanja:

- **aktivna** (angl. active) oz. aktivnost v izvajanju,
- **v premoru** (angl. on pause) oz. prekinjena,
- **ustavljena** (angl. stopped),
- **ugasnjena** (angl. shut down),

o katerih je razvijalec obveščen preko metod:

- **onCreate(Bundle) –** metoda je klicana ob zagonu aktivnosti,
- **onStart() –** aktivnost bo prikazana uporabniku,
- **onResume() –** aktivnost je pripravljena na interakcijo z uporabnikom,
- **onPause()** – metoda se proži tik preden gre v ozadje,
- **onStop()** – aktivnost ni več vidna uporabniku,
- **onRestart()** – aktivnost bo ponovno prikazana na zaslonu,
- **onDestroy() –** aktivnost je klicana tik preden je ugasnjena,
- **onSaveInstanceState(Bundle) –** metoda za shranjevanje stanja aktivnosti preden je ta ugasnjena,
- **onRestoreInstanceState(Bundle) –** klicana metoda, ko je bila aktivnost ponovno klicana iz predhodno shranjenega stanja.

Ohranjanje stanja nekaterih aktivnosti je zelo uporabna zadeva. Pri vračanju nazaj na neko želeno aktivnost, si želimo aktivnost v stanju, kot je bila pred tem. To nam omogoča razred Bundle, ki se nahaja v metodah *onCreate(), onSaveInstanceState(Bundle)* in *onRestoreInstanceState(Bundle)* kot parameter. Razred vsebuje metode za shranjevanje in pridobivanje stanja (*put, get*).
Poleg aktivnosti (*angl. Activities*) imamo pri platformi Android še naslednje aplikacijske komponente:

- Storitve (angl. Services): Komponenta, ki teče v ozadju brez neposredne uporabniške interakcije.
- Ponudniki vsebine (angl. Content Providers): Omogočajo opravljanje z množico podatkov preko posebnega vmesnika. Kot primer je ponudnik vsebine za kontakte.

- Prejemniki oddajanja (angl. Broadcast Receiver): Komponenta se odziva na sporočila sistema ali aplikacije. Na primer sporočilo, da je baterija skoraj prazna, da se bo ugasnil zaslon itd.

Aktivnosti, storitve in prejemnik oddajanja se prožijo z asinhronimi sporočili, imenovanih namen (**intent**). Poganjanje druge aktivnosti implementiramo s kreiranjem novega objekta Intent, ki mu kot parameter podamo trenutno aktivnost ter razred aktivnosti, ki jo želimo prožiti. Na koncu je potreben še klic metode *startActivity(i):*

```
Intent i = new Intent(this, ProženaAktivnost.class);
startActivity(i);
```

4.3 Diagram aplikacije

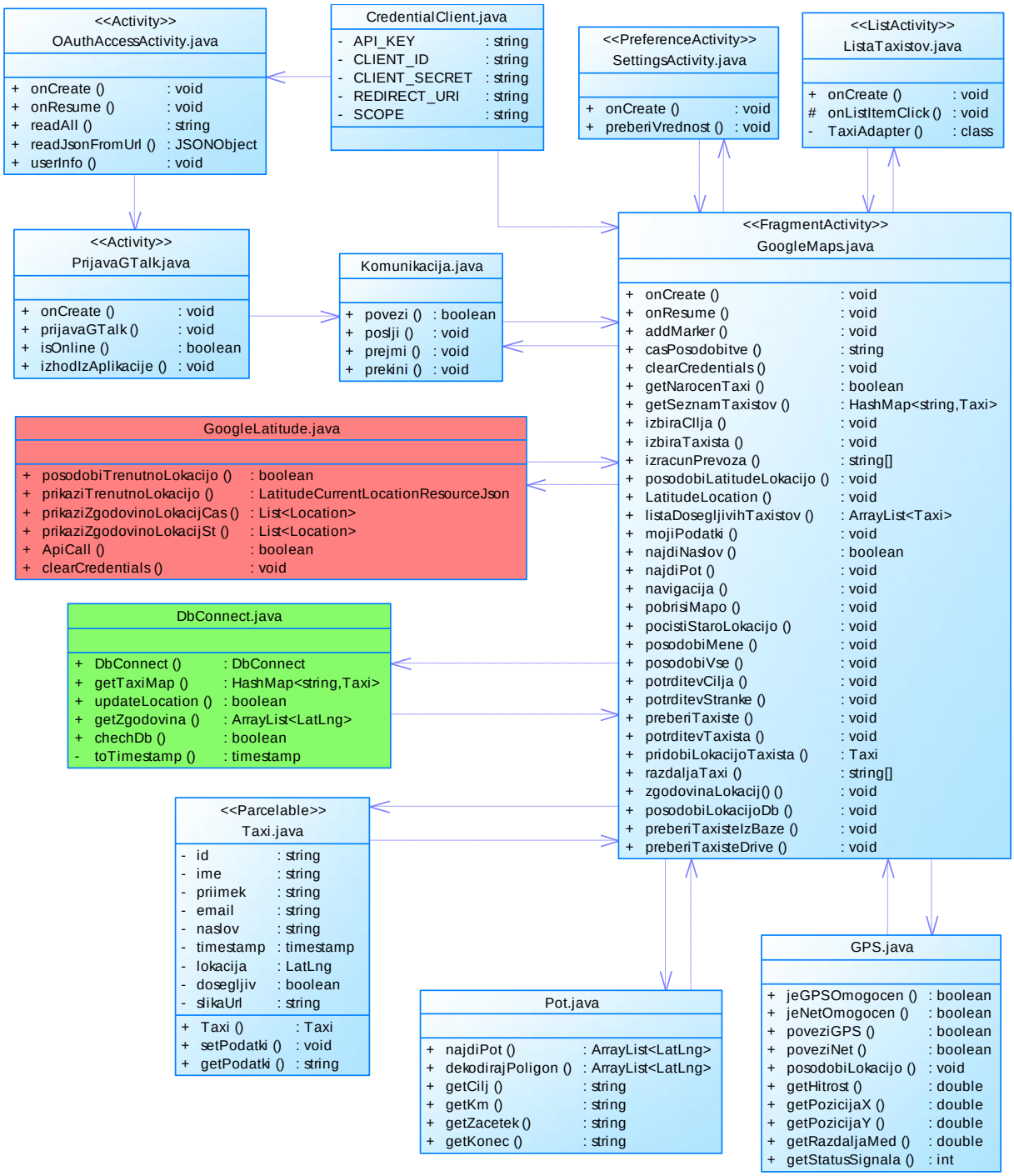
STRANKA	TAKSIST
 Pametni mobilni telefon z operacijskim sistem **Android 2.3 in novejši**.	
Avtentikacija in avtorizacija uporabnika storitev Google s protokolom **OAuth 2.0**. (račun Google) ***Google OAuth2 API v2***	
 Prikaz taksistov na zemljevidu (**Google Maps**) ter trenutne lokacije prijavljenega v aplikacijo. ***Google Maps Android API v2***	
Lokacije taksistov pridobljene preko storitve **Google Latitude**. ***Google Latitude API v1***	
 Pridobivanje vira podatkov o taksistih s storitvijo **Google Drive**. (ID, ime, priimek, email) ***Google Drive API v2***	
Upokojeno storitev **Google Latiude** zamenjamo s podatkovno bazo **MySQL.** Za dostopanje do baze z naprave Android uporabljamo skripte **PHP.**	
Prikaz liste taksistov → Lista je urejena po oddaljenosti taksistov od trenutne lokacije stranke. Podatki taksista: ime, email, slika, oddaljenost, čas prihoda, naslov trenutne lokacije.	
GPS → posodabljanje stranke in taksistov. V primeru slabega GPS signala, lokacijo posodabljamo preko mobilnega omrežja (3G, EDGE)	

<table>
<tr><td colspan="2">Algoritem za izračun oddaljenosti taksista od stranke, izračun oddaljenosti stranke do njenega cilja in izračun cene prevoza. To nam omogoča API za iskanje poti po zemljevidu Google Maps.</td></tr>
<tr><td colspan="2">  Komunikacija s storitvijo Google Talk.</td></tr>
<tr><td colspan="2">Izmenjava podatkov s formatom JSON.</td></tr>
<tr><td>Izbira cilja → želeni naslov ali določitev lokacije na zemljevidu.</td><td>Nastavitev dosegljivosti → v primeru zasedenosti taksista je stranki nedosegljiv in neviden.</td></tr>
<tr><td>Izbira taksista → izpis vseh podatkov taksista. Izbira taksista preko zemljevida ali liste taksistov.</td><td></td></tr>
<tr><td rowspan="2">Uporabniku »stranka« je ta funkcionalnost onemogočena.</td><td>Posodabljanje lokacije v storitev Google Latitude.</td></tr>
<tr><td>Posodabljanje lokacije v podatkovno bazo MySQL.</td></tr>
<tr><td rowspan="2">Uporabniku »stranka« je ta funkcionalnost onemogočena.</td><td>Zgodovina lokacij taksista preko storitve Google Latitude.</td></tr>
<tr><td>Zgodovina lokacij preberemo iz podatkovne baze MySQL.</td></tr>
</table>

Diagram aplikacije nam nazorno prikazuje vse funkcionalnosti s strani različnih vlog uporabnikov (stranka ali taksist). V rdečem okvirju so prikazane funkcionalnosti, ki smo jih zaradi nepričakovane upokojitve storitve Google Latitude morali zamenjati z novimi delujočimi rešitvami. Nova rešitev je realizacija podatkovne baze MySQL.

4.4 Razredi aplikacije

Za dobro infrastrukturo obširne mobilne aplikacije, smo najprej izdelali razredni diagram. Temu pravimo objektno načrtovanje, za lažje predstavljanje in razumevanje poteka razredov. Uporaba objektno usmerjenih tehnik, pri analizi in načrtovanju, občutno zmanjša vpliv sprememb na razvoj celotnega sistema. Tudi, če se spremenijo zahteve, je potrebno narediti spremembe le na določenem mestu v razredu, ali preprosto dodati še kakšen podrazred. Razredni diagram prikazuje statično strukturo modela sistema. Prikazuje razrede, strukturo razredov, metode, atribute ter relacije med razredi. Zaradi boljše razumljivosti so rdeče obarvani razredi tisti, ki so bili zaradi težav opuščeni. Vpeljana je bila nova rešitev oz. nov razred, ki je v diagramu obarvan zeleno.



Slika 07: Prikaz razrednega diagrama prototipa.

Vsak razred ima svojo ključno funkcijo v povezavi z drugimi razredi. Ker je projekt zasnovan za mobilno aplikacijo platforme Android, je večina razredov *Activity*. To so aktivnosti, ki so izmed aplikacijskih komponent najbolj uporabljene. Poznamo še storitve (angl. Services), ponudnike vsebine (angl. Content Providers) in prejemniki oddajanja (angl. Broadcat Receiver). Glavna značilnost aktivnosti je, da imajo svojo življenjsko dobo ter predstavljajo en ekran uporabniškega vmesnika. V nadaljevanju so predstavljeni vsi ključni razredi oz. aktivnosti našega prototipa.

4.4.1 Aktivnost avtorizacije z računa Google

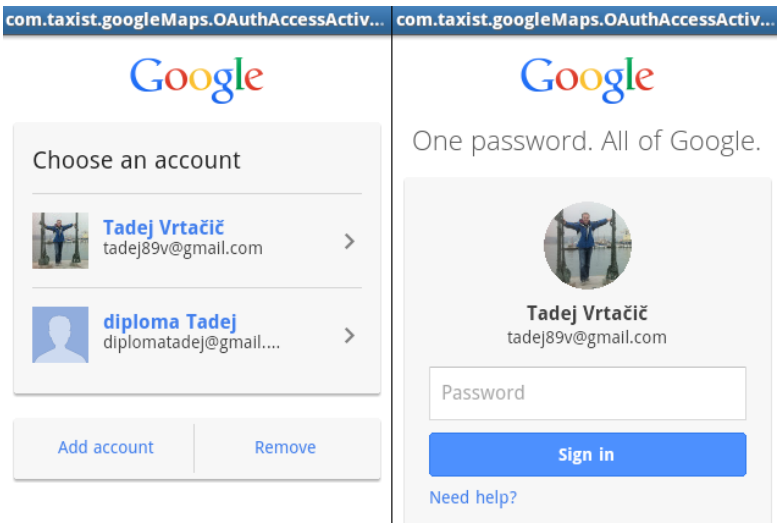
Za uporabo mobilne aplikacije najprej potrebujemo račun Google, ki nam bo omogočal uspešno prijavo v aplikacijo in nadaljnjo uporabo storitev. Dandanes je ustvarjen račun Google že nekaj samoumevnega, saj ponuja obilo delujočih rešitev.

Za avtentikacijo in avtorizacijo aplikacije uporabljamo popularen protokol OAuth 2.0. Njegove ključne funkcionalnosti protokola so uporabljene ločeno od drugih razredov in aktivnosti. Razred »OAuthAccessActivity.java«, torej skrbi za avtorizacijo aplikacije.

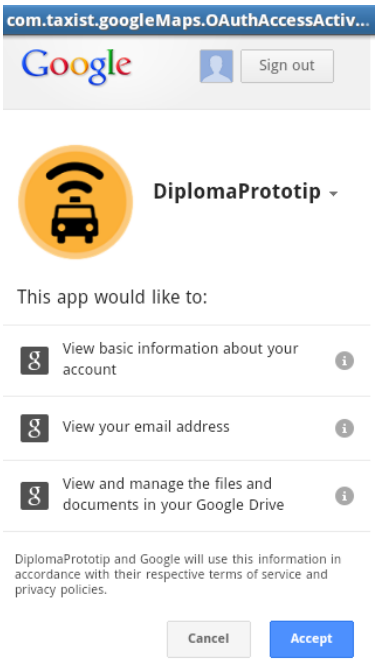
Glavna funkcija razreda je, pridobiti vse parametre in dovoljenja, ki nam bodo omogočala uporabljati aplikacijo z vsemi storitvami Google, ki nam jih le-ta ponuja.
Ključni parametri, ki jih uporabljamo v razredu:

- **Avtorizacijski url** v katerem podamo vse parametre pridobljene z konzole Google APIs.
- **Avtorizacijsko kodo**, za pridobivanje žetona za dostop do storitev.
- **Žeton za dostop in žeton za osveževanje**, ki nam omogočata dostop do Google APIs.

Ob uspešni prijavi uporabnika je potrebna še potrditev uporabe storitev Google. Če uspešno pridobimo vse parametre, ki nam omogočajo pravilno uporabo storitev, se nam pojavi potrditvena stran. Ko uporabnik potrditi dostop do storitev Google je aplikacija pripravljena za uporabo. Vse ključne parametre pri prijavi si aplikacija shrani, tako da ob naslednji uporabi avtorizacija nebo potrebna. Razen v primeru odjave iz našega računa Google.



Slika 08: Izbira in prijava računa Google.



Slika 09: Potrditev dostopa API do želenih informacij.

Ob prvi prijavi v našo aplikacijo moramo zagotoviti tudi prijavo v storitev Google Talk, ki v aplikaciji poteka ločeno od ostalih storitev. To nam omogoča aktivnost »PrijavaGTalk.java«, ki poskrbi, da komunikacija prek storitve Google Talk poteka brez napak. Komunikacija je ključnega pomena, saj je le tako aplikacija funkcionalna s strani uporabnika. V tej aktivnosti preverjamo tudi prisotnost uporabnika v PB. Tako lahko razberemo za katero vrsto uporabnika gre, taksista ali stranko taksistov. Spodaj je prikazan uporabniški vmesnik aktivnosti »PrijavaGTalk.java«.

Slika 10: Uporabniški vmesnik aktivnosti »PrijavaGTalk.java«.

4.4.2 Osrednja aktivnost - zemljevid Google Maps

Za uporabo zemljevida Google Maps v aktivnosti projekta je najprej potrebno zagotoviti vse knjižnice, ki jih storitev za normalno delovanje uporablja. V projekt namestimo Google Play SDK, ki je skupek knjižic, primerov in dokumentacije. Z namestitvijo tega nabora orodij pridobimo knjižnični projekt »google-play-services_lib«, ki ga vključimo v naš projekt in s tem omogočamo nabor vseh knjižnic, ki jih potrebujemo. Pri razvoju projekta z zemljevidom Google Maps potrebujemo aplikativni ključi (API key), ki ga pridobimo z registracijo projekta v konzoli Google APIs. V konzoli je potrebno omogočiti dostop do uporabniškega vmesnika *Google Maps Android API v2*.

Glavna aktivnost naše aplikacije »GoogleMaps.java« je aktivnost, ki za postavitev uporabniškega vmesnika vsebuje zemljevid Google Maps. V postavitvi UI (angl. User Interface) uporabljamo gradnik »fragment«, s katerim v aplikaciji prikazujemo zemljevid. Fragment je del aktivnosti, ki ima svoj življenjski cikel ter prejema svoje vhodne dogodke. Te lahko dodamo in odstranimo med tekočo aktivnostjo oz. dejavnostjo (podobno kot pod-aktivnosti, ki jih lahko ponovno uporabljamo v različnih aktivnostih).

Primer uporabe gradnika »fragment« v razporeditvi ekrana naše glavne aktivnosti:

```
<fragment
android:id="@+id/map"
android:name="com.google.android.gms.maps.SupportMapFragment"
     android:layout_width="match_parent"
android:layout_height="match_parent" />
```

Na UI aktivnosti z zemljevidom prikazujemo podatke, ki so ključni za storitev prevoza, ki jo taksist nudi koristnikom. Obnaša se različno glede na vlogo uporabnika aplikacije. Aktivnost omogoča naslednje pomembne funkcionalnosti:

- **Posodabljanje in prikaz naše trenutne lokacije.**

Za posodobitev naše trenutne lokacije uporabljamo GPS (angl. Global Positioning System). Z globalnim sistemom pozicioniranja (GPS) določimo točno lego in čas kjerkoli na Zemlji. Ob primeru slabega signala GPS našo trenutno lokacijo pridobimo z mobilnim omrežjem ali brezžičnim omrežjem WiFi, ki ga dandanes lahko brezplačno uporabljamo v večjih mestih. V našem projektu nam nalogo posodabljanja naše trenutne lokacije opravi razred »GPS.java«. V razredu najprej preverjamo ali je senzor GPS našega mobilnega telefona vključen, prav tako za omrežje. Nato glede na ponudnika (GPS, omrežje) v aplikaciji generiramo poslušalca. Ta nam ob zaznavanju spremembe naše trenutne lokacije posreduje točno lokacijo. Poslušalec ponudnika lokacije nam vrača ključne parametre: zemljepisna širina in dolžina, čas, hitrost, status signala. Glede na status signala posameznega ponudnika razred naredi preklop na drugega ponudnika. Če na primer izgubimo signal GPS, nam aplikacija poskrbi, da pridobi

trenutno lokacijo uporabnika s pomočjo omrežja. V primeru izgube signala obeh ponudnikov, aplikacija ne deluje pravilno in nas o tem opozori.

Primer preverjanja vključenega senzorja GPS:

```
public boolean jeGPSomogocen() {
     if (lokacijskiManagerGps
               .isProviderEnabled(LocationManager.GPS_PROVIDER))
               return true;
     else
          return false;
}
```

Primer povezave GPS:

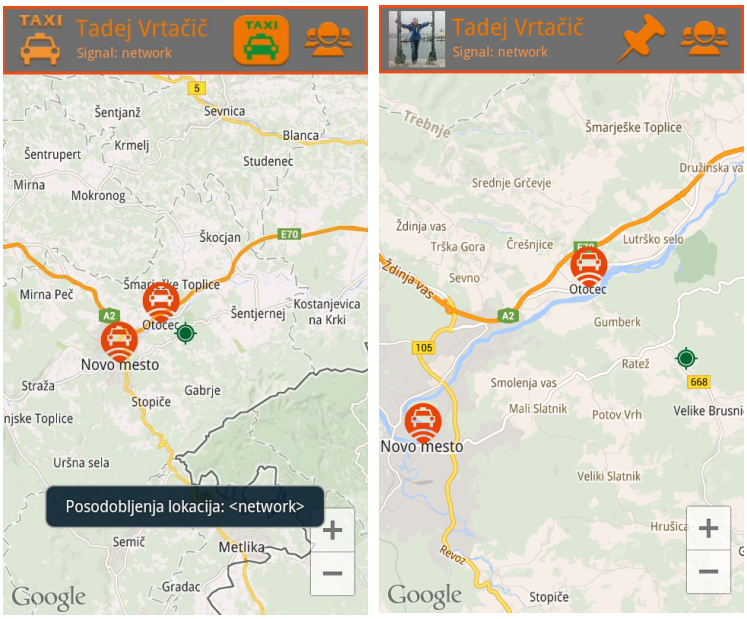
```
private LocationManager lokacijskiManagerGps = null;
public boolean poveziGPS() {
     if (jeGPSomogocen() && zagonGPS == false) {
          LocationListener lokacijskiListener = new
LokPoslusalecGPS();
          lokacijskiManagerGps.requestLocationUpdates(
               LocationManager.GPS_PROVIDER, 0, 0,
lokacijskiListener);

          GPS.zagonGPS = true;

          return true;
     } else
          return false;
}
```

Poslušalec lokacije preko GPS, je implementiran v razredu »LokPoslusalecGPS()«:

```
private class LokPoslusalecGPS implements LocationListener {

     @Override
     public void onLocationChanged(Location location) {
     //Tukaj posodobimo lokacijo
     }
     @Override
     public void onProviderDisabled(String provider) {
          //v primeru izključitve ponudnika
     }
     @Override
     public void onProviderEnabled(String provider) {
          //v primeru vključitve ponudnika

     }
     @Override
     public void onStatusChanged(String provider, int status, Bundle
extras) {
          //z metodo pridobimo status signala
     }
}
```

- **Prikaz vseh dosegljivih taksistov v realnem času.**

Vse ključne podatke o taksistih pridobimo iz baze podatkov, ki se hranijo na strežniku. Podatki se shranijo v strukturirani obliki v zbirki podatkov v aplikaciji. Za shranjevanje podatkov v aplikaciji uporabljamo eno od najbolj popularnih zbirk podatkov v Javi, razred *HashMap*. Ta nam omogoča shranjevanje objektov s ključi, kar omogoča lažje in hitrejše iskanje. Ključ posameznega taksista v zbirki je njegov elektronski naslov. V zbirki podatkov shranjujemo objekt »Taxi«, ki je implementiran v razredu »Taxi.java«. Razred vsebuje vse ključne spremenljivke ter metode za branje in shranjevanje podatkov. To so podatki taksista, ki zajemajo id, ime, priimek, naslov, lokacijo, čas posodobitve lokacije, slika, dosegljivost, oddaljenost in čas prihoda. Vse te informacije, ki se nahajajo v zbirki podatkov aplikacije, pa ne pridobi želena stranka le iz baze podatkov. Ob zagonu programa se poženejo še drugi algoritmi aplikacije, ki omogočajo informacije glede na trenutno lokacijo stranke. Te informacije so oddaljenost taksista od stranke v kilometrih in minutah, ne samo lokacija na zemljevidu ampak tudi točen naslov. Vse te informacije o taksistih, njihovim strankam omogočajo, da so o njih dobro obveščeni. Tako se lahko lažje in boljše odločijo za ponudnika storitve prevoza.



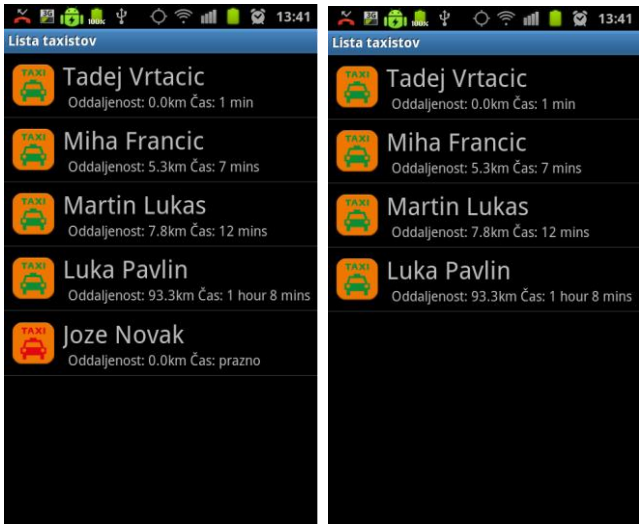
Slika 11: Glavna aktivnost uporabnika taksista (levo) in stranke (desno).

- **Izbira želenega taksista na zemljevidu.**

Naročanje taksista v aplikaciji je dokaj enostavno. Preprosto kliknemo na ikono želenega taksista na zemljevidu ter ga v prikazanem dialogu naročimo. Počakati je potrebno le na potrditev taksista. Spodaj so za boljše razumevanje prikazane slike izbire želenega taksista.

Slika 12: Naročilo stranke izbranega taksista.

Mobilna aplikacija pa ne omogoča prikaz taksistov le na zemljevidu Google Maps aktivnosti »GoogleMaps.java«, ampak jih prikaže na urejenem seznamu aktivnosti »ListaTaxistov.java«. Aktivnost prikazuje seznam vseh dosegljivih taksistov, urejenega od najbližjega pa do najbolj oddaljenega taksista, glede na trenutno lokacijo stranke.



Slika 13: Lista taksistov na strani taksista ali stranke.

- **Izbira cilja prevoza stranke taksi službe na zemljevidu**

Aplikacija je zasnovana tako, da je stranka pred naročanjem taksista primorana določiti svoj cilj prevoza. Tako se lahko taksist na podlagi cilja prevoza odloči, ali bo stranki ugodil in ji nudil prevoz. Določitev cilja je dokaj enostavna. Izbira za nastavitev cilja se nahaja v zgornjem meniju aplikacije. Ob pritisku na izbiro se uporabniku (stranki) prikaže dialog, kjer lahko vpiše naslov cilja, ali pa ga določi kar na zemljevidu.

Slika 14: Primer izbire cilja stranke v aplikaciji.

- **Osnovne nastavitve funkcij zemljevida Google Maps.**

V aplikaciji lahko izvajamo osnovne nastavitve nad zemljevidom, ki nam jih ponuja. Ena izmed njih je nastavitev tipa zemljevida (normalni, satelitski, itd.). Nastavljamo lahko prikaz kontrol zemljevida, kot so kontrole za povečavo in pomanjšavo (zoom) ter kompas zemljevida.

- **Izračun poti do želenega cilja stranke in prikaz na zemljevidu.**

Ob uspešni komunikaciji uporabnikov aplikacije (taksist - stranka) nam aplikacija omogoči izračun poti do vnesenega cilja stranke. To olajša delo taksistu, saj mu na zemljevidu prikaže pot do stranke in do njenega cilja. Algoritem za izračun poti je implementiran v razredu »Pot.java«. Pri tem si pomagamo z eno od funkcionalnosti storitve Google Maps. Ta nam ponuja vsa potrebna navodila za pot (angl. directions), ki jih potrebujemo. Potrebno je vnesti vse parametre, ki jih potrebuje za pravilno delovanje. Z njimi določimo začetek in konec naše poti (preprosto z zemljepisno širino in dolžino) ter način prevoza. Navodila preberemo preko URL-ja:

```
http://maps.googleapis.com/maps/api/directions/json?
                                    origin="začetek poti"
                                    &destination="konecpoti"
                                    &sensor=false
                                    &mode=driving"
```

Parameter »mode« ima nastavljeno vrednost »drive«, kar pomeni, da bomo našo želeno pot prevozili. Povezava nam s pravilnimi parametri vrne podatke v formatu JSON. Podatki nam opisujejo korake celotne poti, kar prikazujemo na zemljevidu Google Maps. Struktura pridobljenih podatkov JSON:


```
{
   "routes" : [{
         "bounds" : {
            "northeast" : {"lat","lng"},
            "southwest" : {"lat","lng"}
         },
         "copyrights" : "Map data ©2013 Google",
         "legs" : [{
               "distance" : {"text","value"},
               "duration" : {"text","value"},
               "end_address",
               "end_location" : {"lat","lng"},
               "start_address",
               "start_location" : {"lat","lng"},
               "steps" : [{
                     "distance" : {"text","value"},
                     "duration" : {"text","value"},
                     "end_location" : {"lat","lng"},
                     "html_instructions",
                     "polyline" : {"points"},
                     "start_location" : {"lat","lng"},
                     "travel_mode"
                      }],
               "via_waypoint" : []
            }],
         "overview_polyline" : {"points"},
         "summary",
         "warnings" : [],
         "waypoint_order" : []
      }],
   "status" : "OK"
}
```

Spodaj prikazani algoritem nam omogoča dekodiranje podatkov iz JSON objekta »polyline«, ki vsebuje točke poti, katere izrišemo na zemljevidu.

```
private ArrayList<LatLng> dekodirajPoligon(String encoded)
{
      ArrayList<LatLng> poly = new ArrayList<LatLng>();
      int index = 0, len = encoded.length();
      int lat = 0, lng = 0;
      while (index < len) {
            int b, shift = 0, result = 0;
            do {
                  b = encoded.charAt(index++) - 63;
                  result |= (b & 0x1f) << shift;
                  shift += 5;
            } while (b >= 0x20);
            int dlat = ((result & 1) != 0 ? ~(result >> 1):(result >>
1));
            lat += dlat;
            shift = 0;
            result = 0;
            do {
                  b = encoded.charAt(index++) - 63;
                  result |= (b & 0x1f) << shift;
                  shift += 5;
            } while (b >= 0x20);
            int dlng = ((result & 1) != 0 ? ~(result >> 1):(result >>
1));
            lng += dlng;

            LatLng position = new LatLng(lat / 1E5, lng / 1E5);
            poly.add(position);
      }
return poly;
}
```

Slika 15: Prikaz informacij storitve prevoza (oddaljenost in čas prihoda taksista, cena).

4.4.3 Razred Google Latitude

Uporaba storitve Google Latitude je bila v aplikaciji dokaj enostavna in funkcionalna, vse dokler storitve niso upokojili. Spodaj opisujemo razred »GoogleLatitude.java«, ki smo ga predhodno uporabljali v prototipu aplikacije.

Odobren dostop do storitev Google nam omogoča uporabo storitve Google Latitude. Google Latitude API potrebuje za dostop do njegovih metod in vseh funkcionalnosti parametre, ki jih pridobimo s protokolom OAuth 2.0.

Najprej implementiramo konstruktor razreda Latitude, ki ga v projektu uporabljamo s pomočjo knjižnice za omenjeno storitev. Primer implementacije:

```
Latitude latitude = new Latitude(transport, access, jsonFactory);
                latitude.apiKey = CredentialsClient.API_KEY;
```

Ob uspešnem klicu Google Latitude API lahko uporabljamo vse funkcionalnosti storitve. Seveda je uporaba storitve odvisna tudi od verzije knjižnice, ki jo v samem projektu pri razvoju uporabljamo. Pozorni moramo biti, da gre za najnovejšo različico, ki jo lahko pridobimo na uradni strani Google za razvijalce mobilnih aplikacij. Uporaba najnovejše različice knjižnice storitve, ki jo uporabljamo, nam omogoča uporabo vseh funkcionalnosti, ki jih ponuja. V razredu »GoogleLatitude.java« uporabljamo naslednje metode oz. funkcionalnosti storitve:

- **Nastavitev parametrov, za uporabo storitve Latitude.**

Po uspešni avtorizaciji aplikacije, s protokolom OAuth 2.0, pridobimo žeton dostopa. Tega, poleg aplikativnega ključa iz konzole Google, uporabimo pri klicu Google API. Pri napačnem vnosu parametrov v klic API-ja bo dostop onemogočen.

- **Samodejno posodabljanje lokacije.**

Namen uporabe samodejnega posodabljanje lokacije v storitvi nam omogoča, da lahko tako taksisti kot koristniki aplikacije vidijo vse prijavljene taksiste, kje se trenutno nahajajo. Posodabljamo jo preko metode *insert(LatitudeCurrentlocationResourceJson),* v kateri imamo parameter razreda *LatitudeCurrentlocationResourceJson.class.* Ta razred nam omogoča, da preko njega v storitev vstavimo vse potrebne podatke o viru lokacije. Glavni trije viri lokacije so vrednost zemljepisne širine in dolžine ter časovni žig. Časovni žig je eden ključnih podatkov, saj nam sporoča kdaj je bila nazadnje lokacija posodobljena v storitev. Primer vstavljanja trenutne lokacije v storitev Latitude:

```
LatitudeCurrentlocationResourceJson
latitudeCurrentlocationResourceJson = new
LatitudeCurrentlocationResourceJson();

latitudeCurrentlocationResourceJson.put("latitude", x);
latitudeCurrentlocationResourceJson.put("longitude", y);
latitudeCurrentlocationResourceJson.put("timestampMS", d);

latitude.currentLocation.insert(latitudeCurrentlocationResourceJson)
.execute();
```

- **Branje zgodovine lokacij uporabnika.**

Zgodovina lokacij določenega uporabnika je uporabna zadeva. Namreč ob koncu dneva, tedna ali meseca si lahko uporabnik ogleda svoje lokacije, kjer se je v določenem obdobju nahajal. To lahko uporablja za svojo statistiko, radovednost ali za katero drugo uporabno zadevo. Zgodovino preberemo z razredom *Location.class* in sicer z naslednjo implementacijo:

```
Latitude.Location.List list = latitude.location.list();
          list.maxTime = cas;
          LocationFeed locationFeed = list.execute();
          List<Location> locations = locationFeed.items;
```

Primer pridobivanja zgodovine lokacij s parametrom »cas«. Ta določa kako daleč (časovno gledano), preberemo zgodovino lokacij od tistega trenutka.

```
Latitude.Location.List list = latitude.location.list();
          list.maxResults = stLokacij;
          LocationFeed locationFeed = list.execute();
          List<Location> locations = locationFeed.items;
```

V tem primeru pa preberemo zgodovino lokacij glede na število lokacij, ki jih določimo s parametrom »stLokacij«.

- **Branje trenutne lokacije uporabnikov.**

Branje svoje trenutne lokacije je možna z metodo get(), ki se nahaja v razredu CurrentLocation.class in je podrazred razreda Latitude.class. Lokacijo pridobimo s spodaj prikazano implementacijo.

```
LatitudeCurrentlocationResourceJson latitudeCurrentlocation =
latitude.currentLocation.get().execute();
```

Vrednost, ki jo vrne Google API z uporabo zgornje implementacije, je tipa LatiudeCurrentlocationResourceJson. S tem razredom pridobimo vse podatke o lokaciji. Vendar se pojavi problem, ko želimo pridobiti lokacijo nekega drugega uporabnika. To nam že omenjeni razred »Latitude«, z uporabljeno najnovejšo knjižnico v projektu, ne omogoča. Zadeve se je potrebno lotiti na drugačen način. Namreč storitev Google Latitude uporablja za dostop do lokacij drugih uporabnikov posebno spletno aplikacijo, in sicer uporabnik lahko v storitvi Latitude omogoči vidnost trenutne lokacije vsem. Temu pravimo javna značka lokacije. To lokacijo pridobijo drugi s pomočjo omenjene spletne aplikacije. Do aplikacije dostopamo z uporabnikovim identifikacijsko številko (ID) storitve. ID je različen od osnovnega ID računa Google.

Primer URL-ja spletne aplikacije s zahtevanimi parametri:

```
http://latitude.google.com/latitude/apps/badge/api?user=-
16990566678674667 73&type=json
```

V zahtevanem URL-ju podamo poleg uporabniškega ID-ja še tip želenega izpisa podatkov. Uporabljamo format podatkov JSON, katero strukturo pridobljenih podatkov uporabljamo skozi celotno aplikacijo.


```
{
"type": "FeatureCollection",
"features": [
{
    "type": "Feature",
    "geometry": {"type": "Point", "coordinates": [15.1591167, 45.7946897]},
    "properties":
    {
        "id": "-16990566678674667 73",
        "accuracyInMeters": 25,
        "timeStamp": 1373625666,
        "reverseGeocode": "8000 Novo mesto, Slovenija",
        "photoUrl": "http://latitude.google.com/latitude/apps/badge/api?type=photo
        "photoWidth": 96,
        "photoHeight": 96,
        "placardUrl": "http://latitude.google.com/latitude/apps/badge/api?type=pho
        "placardWidth": 56,
        "placardHeight": 59
    }
}]
}
```


Slika 16: Izpis podatkov formata JSON javne lokacije storitve Latitude.

Uporaba Google Latitude v aplikaciji je zahtevala shrambo liste taksistov, ki uporabljajo to storitev. Te smo zagotovili s svojim seznamom vseh taksistov, ki uporabljajo omenjeno storitev. Seznam taksistom smo zaradi dobre funkcionalnosti oziroma vseskozi spreminjanja seznama vseh taksistov shranili na strežniku. Začetna ideja je bila, da bi shrambo taksistov vsebovala kar aplikacija, kar nebi omogočalo dinamičnost. Onemogočeno nam bi bilo

spreminjanje ali dodajanje novih taksistov. Shrambo na spletu nam je omogočala storitev Google Drive. Tu smo shranjevali seznam taksistom v JSON formatu, ki smo ga lahko poljubno dodajali ali spreminjali. V datoteki formata JSON so podani ključni podatki taksista. Eden od njih je identifikacijska številka uporabnika storitve Google Latitude, s katero pridobimo javno lokacijo želenega uporabnika, v našem primeru taksista. S tem iz drugih storitev oziroma virov preberemo želene podatke lokacij vseh taksistov. V nadaljnjem razvoju smo zaradi upokojitve Google Latitude opustili tudi to rešitev. S svojo novo alternativo podatkovne baze MySQL bi bila uporaba Google Drive nesmiselna. Hranjenje seznama taksistov smo tako zagotovili s svojo PB. S tem smo izboljšali učinkovitost aplikacije, saj vse podatke shranjujemo in beremo z enega mesta, mesta podatkovne baze MySQL. Za razumevanje razvoja in delovanja predhodnega prototipa je spodaj opisan postopek implementacije storitve Google Drive.

Najprej moramo poskrbeti za registracijo Google Drive API v našem projektu. Omogočimo ga v konzoli Google, kjer pridobimo potrebne parametre za nadaljnjo uporabo API-ja v našem projektu. V projekt moramo nato še vključiti vse potrebne knjižnice, ki jih potrebujemo za pravilno delovanje API-ja. Omogočiti je potrebno knjižnico odjemalca *Google Play Services*. Datoteka *AndroidManifest.xml*, ki predstavlja bistvene podatke aplikacije mora še vsebovati naslednje vrstice programske kode:

```
<uses-permission android:name="android.permission.GET_ACCOUNTS"/>
<uses-permission android:name="android.permission.INTERNET" />
```

Glavna funkcionalnost razreda je, da pridobimo datoteko, v kateri je shranjen seznam vseh taksistov. Datoteko preberemo z njeno identifikacijsko številko (ID). Ta je zapisana v aplikaciji kot privatna spremenljivka in se ne spreminja. Datoteka je javna, zato lahko do nje s pomočjo ID-ja dostopajo vsi uporabniki. V vsebini datoteke je nabor vseh taksistov. Vsak taksist vsebuje ime in priimek, elektronski naslov in svojo ID storitve Google Latitude. To so ključni podatki, ki se jih uporablja za nadaljnjo uporabo. S pomočjo elektronskega naslova Google pridobimo vse podatke o računu taksistu (slika, številka, itd). ID taksista pa nam omogoča pridobitev trenutne javne lokacije iz storitev Google Latitude. To lokacijo taksist ob svoji dosegljivosti (nezasedenosti) sprotno posodablja. Če se datoteka na datotečni storitvi Google Drive spremeni, so ob vsaki prijavi ob aplikaciji ponovno prebere najnovejša datoteka oz. nabor vseh taksistov.

Podatki datoteke *taksisti.json* shranjeni na Google Drive:

```
{
"taxisti":[{
      "userId":"5957404823456128006",
      "email":"primer@gmail.com",
      "ime":"Taxi123"
      },
      {
      ...
      }]
}
```

Dostop do zgornje datoteke nam je omogočen preko naslova:

https://googledrive.com/host/0B8UKYUuS-YFkeUNnU2x6MnN3R0k/taxisti.json

4.4.4 Nastavitve aplikacije – SharedPreferences

ShraredPreferenced je razred v OS Android, s katerim shranjujemo zasebne podatke v paru ključ-vrednost. Razred določa splošen okvir, ki omogoča shranjevanje in pridobivanje parov ključ-vrednost primitivnih podatkovnih tipov. Podatki bojo prisotni tudi po zaključeni uporabniški seji, čeprav je bila vaša aplikacija zaključena.

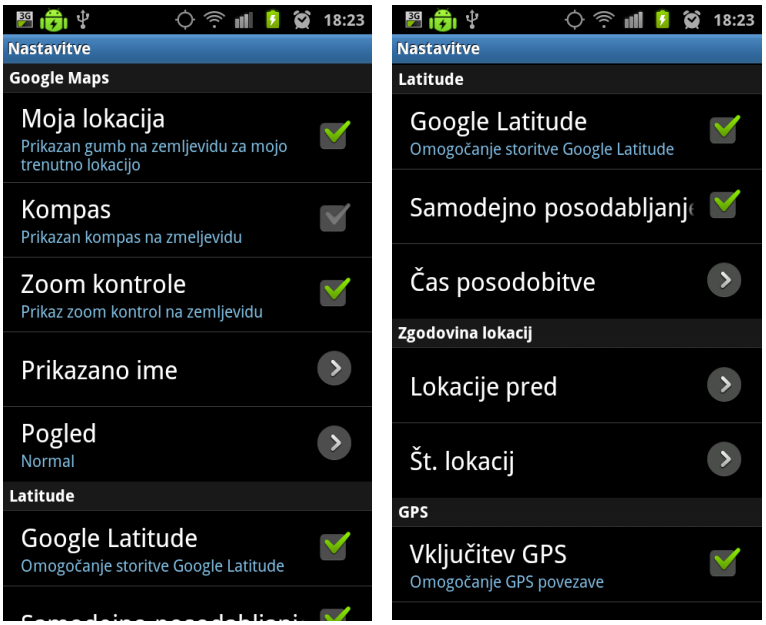
Namenjena je predvsem shranjevanju nastavitev aplikacije. Ko si v aplikaciji nastavimo določeno nastavitev, si jo z razredom *ShraredPreferenced* zapomnimo in shranimo na interni pomnilniški prostor.

Metode razreda SharedPreferences, ki so najbolj pogosteje uporabljene:

- `getSharedPreferences()` – uporabimo, če potrebujemo več nastavitvenih datotek,
- `getPreferenced()` – uporabimo, če potrebujemo le eno datoteko z nastavitvami.

Z zgoraj naštetima metodama lahko preberemo vrednosti in jih nato vsako posebej preberemo po njenih ključih. Preberemo jih z metodo, ki je odvisna od tipa zapisane vrednosti: *getString(ključ), getInt(ključ), getFloat(ključ)…*

Za zapis vrednosti nastavitev ima omenjeni razred, ugnezdeni razred *SharedPrefernces.Editor*, s katerim kličemo želeno metodo. Metodo si izberemo poljubno glede na tip vrednosti, ki jo želimo zapisati: *putString(), putInt(), putFloat(), putBoolean()…* Nastavitve prikazujemo in shranjujemo z aktivnostjo »SettingsActivity.java«.



Slika 17: Nastavitve v naši mobilni aplikaciji.

V aplikaciji razred *SharedPreferenced* ne uporabljamo le za shranjevanje nastavitev aplikacije. Uporabljamo ga tudi za shranjevanje osnovnih podatkov posameznega uporabnika, na samem mobilnem telefonu. Tako omogočimo, da pri naslednjem zagonu aplikacije ne izgubimo vse podatke, ki jih potrebujemo. Podatki se izbrišejo le v primeru odjave iz računa Google, s katerim smo prijavljeni v aplikacijo. Te podatke v aplikaciji shranjujemo, beremo in brišemo z razredom »CredentialSharedPreferences.java«.

Primer shranjevanja podatkov:

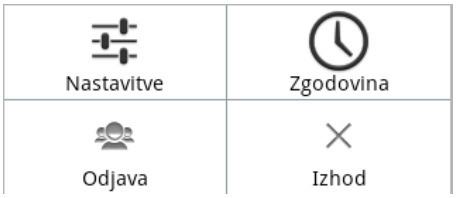
```
private SharedPreferences prefs;
public void writeUser(String googleId, String email, String ime,
String slikaUrl) {
      Editor editor = prefs.edit();

      editor.putString(GOOGLEID, googleId);
      editor.putString(EMAIL, email);
      editor.putString(IME, ime);
      editor.putString(SLIKA_URL, slikaUrl);
      editor.commit();
}
```

Primer brisanja podatkov:

```
public void clearCredentials()
{
      Editor editor = prefs.edit();
      editor.remove(ACCESS_TOKEN);
      editor.remove(EXPIRES_IN);
      editor.remove(REFRESH_TOKEN);
      editor.remove(SCOPE);
      editor.remove(IME);
      editor.remove(GOOGLEID);
      editor.remove(SLIKA_URL);
      editor.remove(VLOGA);
      editor.remove(EMAIL);
      editor.remove(PASSWORD);
      editor.commit();
}
```

4.4.5 Glavni meni

Vsaka mobilna aplikacija naj bi imela kontekstni meni, ki se sproži ob pritisku na gumb »Menu« mobilnega telefona z OS Android. Odvisen je od trenutne aktivnosti oz. od konteksta, zato mu pravimo kontekstni meni. Uporaben je za nastavitve aplikacije in druge osnovne funkcionalnosti, ki jim mora imeti aplikacija. Naša aplikacija v meniju vsebuje odjavo, izhod, nastavitve aplikacije ter zgodovino lokacij prijavljenega uporabnika.



Slika 18: Glavni meni aplikacije.

Neke vrste aktivnega menija, imamo tudi v zgornjem delu postavitvi glavne aktivnosti. Funkcionalnosti tega menija je odvisna od vloge uporabnika, ki je prijavljen v aplikacijo:

- **Vloga taksista.**

V meniju se nam prikaže slika, ime in priimek prijavljenega. Pod imenom se nam izpiše uporaba trenutnega signala senzorjev, ki so nam na voljo za zaznavanje trenutne lokacije. Če na trenutni lokaciji nimamo signala GPS, nam aplikacija samodejno posodablja lokacijo preko mobilnega interneta. Na desni strani nam aktivni meni ponuja dve pomembne funkcionalnosti. Prva funkcionalnost je gumb, ki omogoča zasedenost ali prostost taksista. V primeru zasedenosti, je taksist drugim strankam neviden in ga ne morejo naročiti.

- **Vloga stranke.**

Pri prijavi koristnikov taksistov v aplikacijo nam meni ponuja malo drugačne funkcionalnosti. Prva ključna funkcionalnost (nastavitev zasedenosti pri taksistih), se pri strankah oz. koristnikih taksistov zamenja z izbiro za nastavitev želenega cilja. Cilj lahko določimo na samem zemljevidu, ali z vnosom točnega naslova, ki ga nam aplikacija poišče in izriše na zemljevidu Google Maps.

Kot sem omenil zgoraj ima meni, ki je prikazan na glavni aktivnosti, dve ključni funkcionalnosti. Prva se razlikuje glede na vlogo uporabnika. Druga funkcionalnost aktivnega menija pa je skupna obema vlogama in se skoraj ne razlikujeta. Namreč gre izbiro v meniju, ki sproži listo vseh taksistov. Funkcionalnost se glede na vlogo prijave razlikuje le v tem, da se pri taksistih prikažejo vsi taksisti (tudi zasedeni), pri koristnikih taksistov pa so na listi le dosegljivi taksisti, ki jih lahko izbere za storitev prevoza. S pomočjo algoritma v aplikaciji je lista taksistov urejena po oddaljenosti taksista, od trenutne lokacije uporabnika. Tako ponuja koristnikom možnost izbire taksista, z minimalnim časom čakanja na prevoz.

Slika 19: Aktivni meni taksista ali stranke UI »GoogleMaps.java«.

4.5 Podatkovna baza MySQL

Podatkovna baza je organizirana zbirka logično povezanih podatkov. Načrtovana je tako, da zadovoljuje informacijske potrebe naše aplikacije in je ključnega pomena. Poenostavljeno gledano je podatkovna baza zelo velika shramba prej vnesenih podatkov, ki zadošča določenim pogojem. Pogoji zagotavljajo celovitost podatkov in učinkovitost delovanja. Tako podatkovna baza prispeva k uspešnejšem delu uporabnikov.

V začetku razvoja našega projekta je našo podatkovno bazo in vse strežniške storitve pokrivala storitev Google Latitude ter storitev Google Drive. Podjetje Google je storitev Latitude, 9. avgusta 2013, upokojilo. S tem so onemogočene vse delujoče funkcionalnosti storitve. S svojo alternativo smo to nadomestili in aplikacijo naredili bolj neodvisno in omogočili fleksibilno delovanje, kot je to omogočala storitev Latitude. Kreirali smo podatkovno bazo MySQL.

4.5.1 Realizacija podatkovne baze

Za hitro in fleksibilno delo s podatki je potrebna dobra realizacija podatkovne baze. Najprej moramo poskrbeti za dobro arhitekturo podatkovne baze. Poznamo tri-nivojsko arhitekturo, ki jo sestavljajo:

- **Zunanji nivo** opsuje uporabo podatkov.
- **Konceptualni nivo** opredeljuje celovit pogled vseh vrst podatkov in povezav med podatki v PB.
- **Fizični nivo** opredeljuje fizični način shranjevanja in dostope do podatkov.

Sistem za upravljanje s podatkovnimi bazami (SUPB) je zbirka programov, ki

- omogoča kreiranje nove PB in definiranje njene strukture;
- omogoča učinkovito poizvedovanje in spreminjanje podatkov;
- varuje podatke pred nesrečami in neavtoriziranimi dostopi;
- nadzoruje sočasen dostop večjega števila uporabnikov do podatkov.

Naša aplikacija deluje na principu odjemalec-strežnik. Odjemalec (angl. Client) je naša aplikacija s katero zagotavljamo predstavitev podatkov. Strežnik (angl. Server) nam omogoča obdelavo, povezovanje, storitve baze in primeren vmesnik. Glavni dve značilnosti aplikacij odjemalec-strežnik so:

- Delitev podatkov med več odjemalci in strežnikom, ki nudi storitve vsakemu izmed odjemalcev.
- Komunikacija poteka s pomočjo dobro definiranega niza standardnih programskih vmesnikov (angl. API) in klicev oddaljenih procedur (angl. RPC – Remote Procedure Call).

Najprej je bila naša naloga zagotoviti strežniško plat, kjer smo kreirali podatkovno bazo pod imenom »tadejvrtacic«. Strežnik se nahaja na naslovu *alfa.pip.si*. Splošne značilnosti strežnika so:

MySQL

- Strežnik: Localhost z vtičnikom UNIX.
- Različica strežnika: 5.0.67-Max.
- Verzija protokola: 10.
- MySQL kodna tabela: UTF-8 Unicode.

phpMyAdmin

- Verzija: 3.2.2.
- Uradna domača stran phpMyAdmin.

Web server

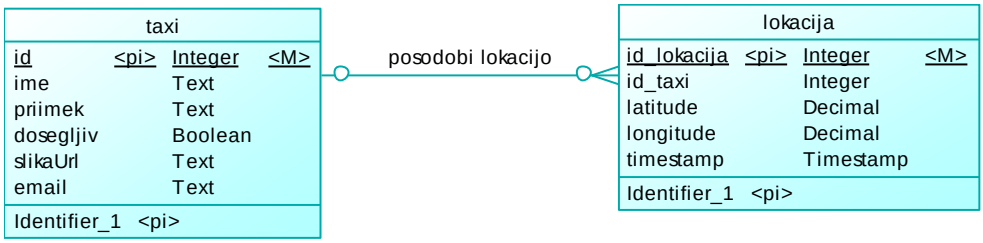
- Apache/2.2.10 (Linux/SUSE).
- Verzija odjemalca MySQL: 5.0.67.
- Razširitev PHP: MySQL.

Načrtovanje podatkovne baze je postopek opredelitve in razvoja strukture PB. Mera za pravilnost načrtovane sheme PB je realni svet. Od tod sledi, da mora vsebina PB odražati podatke, pravila in izjeme iz realnega sveta.

Pri strukturiranju tabel, naše podatkovne baze, smo kreirali dve tabeli. Prva imenovana »taxi« vsebuje atribute, s katerimi shranjujemo glavne podatke o taksistu. V drugi tabeli z imenom »lokacija«, pa shranjujemo lokacije posameznega taksista. Povezavo ali entiteto med tabelama smo zagotovili z ustvarjanjem primarnih in sekundarnih ključev. Tabela »taxi« je s primarnim ključem povezana s sekundarnim ključem tabele »lokacija«, ki ima tudi svoj primarni ključ. Sklicevanje med tabelama smo kreirali s SQL poizvedbo:

```
ALTER TABLE lokacija
ADD FOREIGN KEY (id_taxi)
REFERENCES taxi(id)
```

Relacijska PB je sestavljena iz množice poimenovanih in logično povezanih tabel. Spodaj je prikazan relacijski model podatkov:



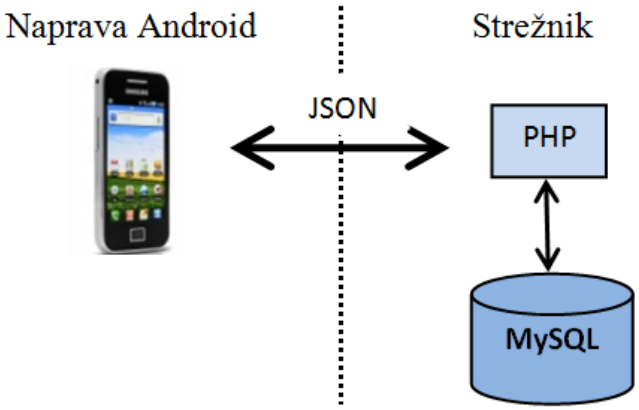
Slika 20: Relacijski model podatkovne baze.

4.5.2 Povezava naprave Android s podatkovno bazo MySQL

Direktna povezava do podatkovne baze MySQL v OS Android ni mogoča. Najbolj razširjen način povezave, z oddaljeno podatkovno bazo MySQL, je uporaba neke vmesne storitve. Ker se MySQL običajno uporablja skupaj s PHP, je to ena od najlažjih rešitev.

Torej smo za povezavo aplikacije Android, s podatkovno bazo MySQL, uporabili PHP skripte, s katerimi upravljamo PB. Skripte PHP poženemo z uporabo HTTP protokola iz sistema Android.

1. Ko se aplikacija Android začne izvajati, se bo naprava Android s protokolom HTTP povezala s skripto PHP.
2. Nato skripta PHP s svojo poizvedbo prebere podatke iz podatkovne baze in jih vrne v kodiranem načinu formata JSON.
3. Android naprava kodirane podatke JSON razčleni in jih uporablja v svoji aplikaciji.



Slika 21: Povezava naprave Android s podatkovno bazo MySQL

V napravi Android z aplikacijo upravljamo s podatki formata JSON, ki jih pridobimo s pomočjo skripte PHP. V samo skripto PHP pa vstavljamo želene SQL poizvedbe, s katerimi pridemo do želenih podatkov v podatkovni bazi.

V našem projektu za nalogo povezovanja mobilne aplikacije s PB, skrbi razred »DbConnect.java«, ki vsebuje metode katere imajo vsaka svojo funkcijo.

- **Metoda chechDb(String email):**

Ob prijavi v aplikacijo se poleg avtorizacije računa Google preveri prisotnost uporabnika v podatkovni bazi. To preverimo z metodo »checkDb()« v razredu »DbConnect.java«. V podatkovni bazi preverimo ali obstaja elektronski naslov, s katerim smo se prijavili v aplikacijo. Tako lahko aplikacija razloči za katerega prijavljenega uporabnika gre. Če je elektronski naslov prisoten v bazi podatkov, gre za taksista in ima drugačne privilegije za delovanje mobilne aplikacije.

Preverjanje v metodi izvedemo s pomočjo skripte PHP, ki se izvaja na strežniku. Skripto v metodi poženemo s protokolom HTTP:

```
HttpClient httpclient = new DefaultHttpClient();
HttpPost httppost = new HttpPost("pot do lokacije skripte PHP na
strežniku");
HttpResponse response = httpclient.execute(httppost);
```

Do skripte na strežniku je potrebno posredovati parameter metode »email«. Tako izvedemo preverjanje prijavljenega uporabnika. Posredujemo ga z naslednjo implementacijo:

```
ArrayList<NameValuePair> nameValuePairs = new
ArrayList<NameValuePair>();
nameValuePairs.add(new BasicNameValuePair("email", email));
httppost.setEntity(new UrlEncodedFormEntity(nameValuePairs));
```

Spodaj je prikazana poizvedba, ki se izvaja na strežniku preko skripte »apiCheck.php«.

```
$email = $_POST['email'];

$query = mysqli_query($con,"SELECT id FROM taxi WHERE
email='$email'");
$row = mysqli_fetch_row($query);
if(!$row)
      echo "0\n";
else
      echo $row[0];
```

Parameter $con v sintaksi je povezava do podatkovne baze. Primer implementacije spremenljivke $con:

```
$con = mysqli_connect("localhost","upor. ime","geslo","ime PB");
```

Skripta nam za uspešno poizvedbo vrne vrednost id in vrednost »0« za neuspešno. Vrednost id iz podatkovne baze uporabimo pri posodabljanju lokacije.

- **Metoda getTaxiMap():**

Ključna vloga podatkovne baze je vsebovanje osnovnih podatkov taksistov (ime, priimek, email) ter podatkov o trenutnih lokacijah (zemljepisna širina, dolžina, čas posodobitve). Metoda »getTaxiMap()« nam poskrbi, da iz podatkovne baze preko skripte PHP pridobi vse te ključne podatke. Skripta »apiGet.php« s poizvedbo:

```
$query = mysqli_query($con, "select t1.*,
                            t2.latitude, t2.longitude, t2.timestamp
                            from taxi t1
                            inner join(
                                  select * from lokacija
                                  order by timestamp desc
                                  )t2
                            on t1.id = t2.id_taxi
                            group by t1.id");
while($row = mysqli_fetch_assoc($query))
{
      $output[]=$row;
}
//vrne podatke v json formatu
print(json_encode($output));
```

Podatke vrne v formatu JSON, ki jih z metodo preberemo in jih shranimo v zbirko podatkov (HashMap) s ključi. Ključi so v zbirki podatkov elektronski naslovi uporabnikov. Z njimi lažje dostopamo do posameznega shranjenega taksista Kot rezultat nam metoda vrne zbirko vseh taksistov, ki se nahajajo v podatkovni bazi MySQL. Te kasneje potrebujemo za nadaljnje pravilno delovanje aplikacije.

Format JSON podatkov, ki ga vrne skripta »apiGet.php« na strežniku:

```
[{
      "id":"6",
      "ime":"Joze",
      "priimek":"Novak",
      "dosegljiv":"0",
      "slikaUrl":"https://link.jpeg",
      "email":"joze.novak@gmail.com",
      "latitude":"12",
      "longitude":"12",
      "timestamp":"2013-09-29 12:19:56"
},
{ … }]
```

- **Metoda updateLocation(String email, double lat, double lon, boolean dos):**

Da aplikacija nudi podatke v realnem času, mora uporabnik aplikacije (taksist) svojo trenutno lokacijo vseskozi posodabljati. Lokacije posodablja z metodo »updateLocation«, s parametri »email«, »lat«, »lon«, »dos«, s katerimi tako posredujemo elektronski naslov, zemljepisno širino in dolžino ter dosegljivost taksista do skripte »apiUpdate.php«, ki se prav tako nahaja na strežniku, s podatkovno bazo MySQL. Poizvedba tako v skripti, s pravilno pridobljenimi parametri, posodobi trenutno lokacijo izbranega uporabnika..

```
$latitude = $_POST['latitude'];
$longitude = $_POST['longitude'];
$id = $_POST['id'];
$dosegljiv = $_POST['dosegljiv'];

$query = mysqli_query($con,"insert into lokacija (id_taxi,
latitude,longitude)
values ($id,$latitude,$longitude)");
$query2 = mysqli_query($con,"update taxi set dosegljiv = $dosegljiv
                             where id = $id");

                             if($query && $query2)
                                echo "1\n";
                             else
                                echo "0\n";
}
```

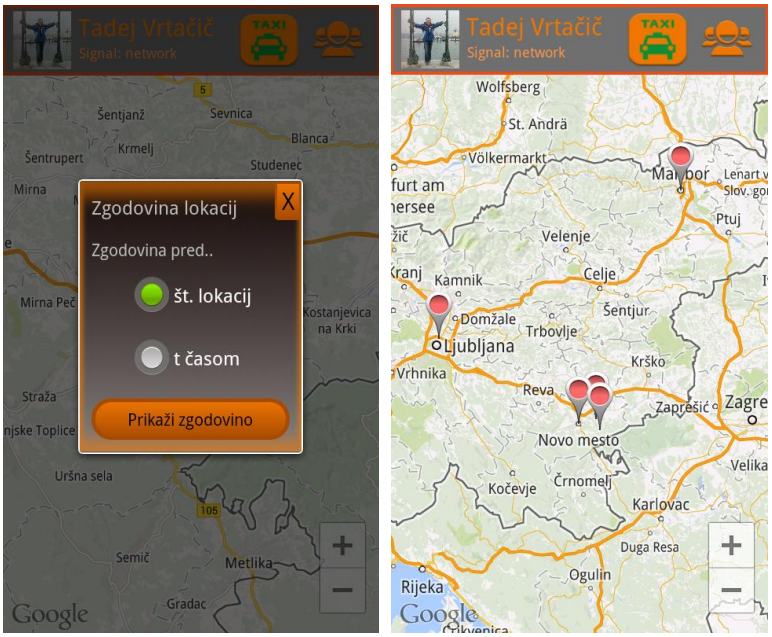
Kot rezultat za uspešno posodobitev podatkovne baze skripta vrne vrednost »1« oz. »true« ali v nasprotnem primeru »0« oz. »false«.

- **Metoda getZgodovina(int userID, String stLokacij, String timestamp)**

Z realizacijo svoje rešitve hranjenja lokacij smo zagotovili tudi eno izmed funkcij, ki jih je ponujala storitev Google Latitude. Z metodo getZgodovina() smo uporabniku aplikacije na strani taksista omogočili vpogled v zgodovino svojih lokacij. Parametra metode »stLokacij« in »timestamp« sta namenjena različnemu vpogledu v zgodovino lokacij. Oba parametra sta nastavljena v nastavitvah aplikacije. Ob izbiri zgodovine v glavnem meniju aplikacije, nam

dialog ponuja dva načina vpogleda: glede na število lokacij ali glede na čas. Aplikacija lokacije je prikazana na zemljevidu Google Maps.

Spodaj nam sliki prikazujeta izbiro vpogleda v zgodovino lokacij in prikaz lokacij na zemljevidu.



Slika 22: Prikaz zgodovine lokacij taksista.

Pridobljene podatke o lokacijah pridobimo pod dvema pogojema: Prvi pogoj je prikaz zadnjih n lokacij. Število lokacij nastavimo poljubno v nastavitvah aplikacije. Želene podatke n lokacij pridobimo s spodnjo poizvedo, ki je del skripte PHP z imenom »apiZgodovina.php«.

```
$id_user = $_POST['id_user'];
$st_lokacij = $_POST['st_lokacij'];
$query = mysqli_query($con, "select latitude, longitude, timestamp
                                                from lokacija
                                                where lokacija.id_taxi =
$id_user
                                                order by timestamp desc
limit $st_lokacij");
          while($row = mysqli_fetch_assoc($query))
          {
                $output[]=$row;
          }
          //vrne podatke v json formatu
          print(json_encode($output));
```

Drugi pogoj pridobitve podatkov iz tabele lokacij PB, je čas. V nastavitvah aplikacije lahko nastavimo vpogled v zgodovino lokacij pred določenim časom. Npr. pred eno uro, enim dnem ali podobno. Ta pogoj smo, tako kot prvega, implementirali v skripti PHP »apiZgodovina.php«, kot prikazuje spodnja programska koda:

```
$id_user = $_POST['id_user'];
$cas = $_POST['cas'];
$query = mysqli_query($con, "select latitude, longitude, timestamp
                                                from lokacija
                                                where lokacija.id_taxi =
$id_user and
                                                      timestamp >
'$cas'");
          while($row = mysqli_fetch_assoc($query))
          {
                $output[]=$row;
          }
          //vrne podatke v json formatu
          print(json_encode($output));
```

Skripta teče na strežniku s PB in vrne podatke o lokaciji. To so podatki zemljepisne širine in dolžine in čas posodobitve lokacije.

4.6 Komunikacija med uporabniki

Glavni cilj predstavljene aplikacije je čim lažja komunikacija med taksistom in stranko. To smo v aplikaciji omogočili z zelo popularno aplikacijo *Google Talk*. Namenjena je komuniciranju s prijatelji in ostalimi. S prijavo računa Google v aplikacijo se ob prijavi povežemo tudi s strežnikom, ki skrbi za komunikacijo med dvema računoma Google. S tem zagotovimo, da uporabnik aplikacije za normalno uporabo le-te potrebuje zgolj račun Google, s tem pa hitro, varno in zanesljivo komunikacijo. Ker komunikacija traja in tudi stane, je to ena izmed hitrejših in poceni komunikacij.

4.6.1 Protokol XMPP

XMPP [21] (angl. Extensible Messaging and Presence Protocol) je odprt komunikacijski protokol za sporočanje, ki temelji na Xml standardu. Tudi Google uporablja pri svojemu produktu Google Talk, za komunikacijo protokol XMPP. Za razvoj aplikacij z neposrednim sporočanjem odjemalcev obstajajo različne knjižnice. Eden od najbolj priljubljenih knjižnic XMPP odjemalca je *Smack API*. To je Java knjižnica, ki omogoča razvijalcem kreiranje IM (angl. Instant Messaging) odjemalcev. Za delo knjižnice z Androidom smo uporabili knjižnico *Asmack*.

Najprej smo v razvojnem okolju Eclipse našemu projektu aplikacije dodali knjižnico *asmack*. Nato je bilo potrebno za komunikacijo z Google Talkom poskrbeti za naslednje ključne točke:

- **povezava s GTalk strežnikom** - za povezavo s strežnikom je potrebno najprej nastaviti vse konfiguracijske parametre:

```
public static final String HOST = "talk.google.com";
public static final int PORT = 5222;
public static final String SERVICE = "gmail.com";
```

Naslednji del kode pa prikazuje uporabo Smack API, za povezavo z XMPP strežnikom:

```
ConnectionConfiguration connConfig = new
ConnectionConfiguration(HOST, PORT, SERVICE);
XMPPConnection connection = new XMPPConnection(connConfig);

try {
  //povezava s strežnikom
  connection.connect();
} catch (XMPPException ex) {
  connection = null;
  //Ne morem se povezati na strežnik
}
```

Razred *XMPPConnection* se uporablja za povezavo s XMPP strežnikom, ki ga določi razred *ConnectionConfiguration*, ki za vzpostavitev povezave s strežnikom uporablja konfiguracijske parametre. Za prekinitev povezave uporabimo metodo *disconnect();*

- **prijava v GTalk strežnik** - v strežnik se prijavimo z uporabniškim imenom in geslom, pri tem uporabimo metodo *login().* Spodnji del kode prikazuje prijavo:

```
connection.login(UPORABNIŠKO IME, GESLO);
//primer: connection.login("abc@gmail.com", "geslo");
```

- **nastavitev uporabniške prisotnosti** - nastavljamo status dosegljivosti;
- **pošiljanje sporočil** - sporočila lahko pošljemo z eno od dveh možnosti:
  - V obliki paketov z metodo *sendPacket(Message msg)* razreda *XMPPConnection*.
  - Kot niz sporočil klepeta (angl. Chat Messages) uporabimo razred Chat ali *Normal*.

V aplikaciji je uporabljena prva oblika pošiljanja sporočil. Primer kode:

```
Message msg = new Message(String to, Message.Type type);
msg.setBody("Kako si?");
connection.sendPacket(msg);
```

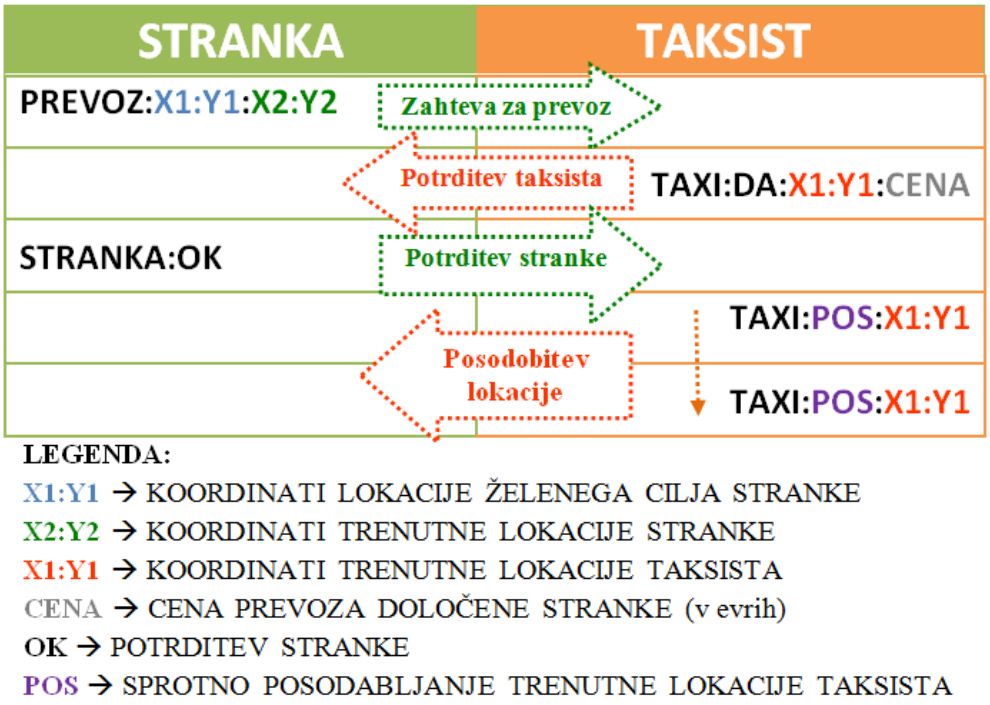
- **prejemanje sporočil** – drugih uporabnikov pridobimo z uporabo:
  - **Mehanizem ankete** zagotovljen z uporabo razreda PacketCollector.
  - **Asinhroni mehanizem** z uporabo razreda PacketListener, kar je priporočljivo.

Koda prikazuje asinhroni način poslušanja dohodnih sporočil.

```
PacketFilter filter = new MessageTypeFilter(Message.Type.normal);
connection.addPacketListener(new PacketListener() {
  public void processPacket(Packet packet) {
    Message message = (Message) packet;
    String body = message.getBody();
    String from = message.getFrom();
  }
}, filter);
```

4.6.2 Diagram komunikacije med vlogami aplikacije

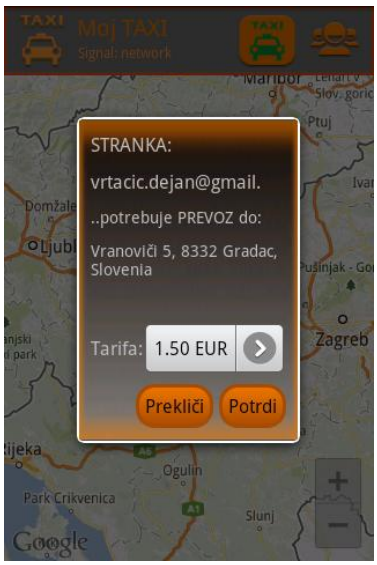
Komunikacija aplikacije s strani stranke in taksista najbolj nazorno prikažemo s spodnjim diagramom.



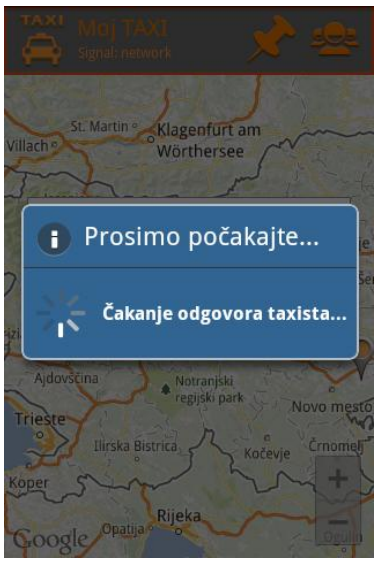
Slika 23: Prikaz komunikacije med stranko in taksistom.

Kot je zgoraj omenjeno naša aplikacija komunicira z drugo s storitvijo Google Talk. Ko si stranka izbere določenega taksista in ga potrdi, aplikacija avtomatično pošlje programsko določen format sporočila. Sporočilo zajema zahtevo za prevoz, s ključno besedo »PREVOZ«, s koordinatami želenega cilja stranke in koordinatami trenutne lokacije stranke. Sporočilo je naslednjega formata: »PREVOZ:X1:Y1:X2:Y2«. Ko je sporočilo poslano na strani stranke, čaka na odgovor z druge strani, to je s strani taksista.

Taksist odobri zahtevan prevoz stranke do določenega cilja z besedo »DA«, v sporočilo se mu avtomatično pripnejo še koordinate njegove trenutne lokacije in izračunan znesek storitve, ki jo bo opravil. Ta storitev je seveda prevoz. Znesek se izračuna po algoritmu, glede na oddaljenost stranke do njenega cilja prevoza. Tarifo na posamezen kilometer prevoza določi taksist sam, ob potrditvi stranke. To prikazuje spodnja slika.

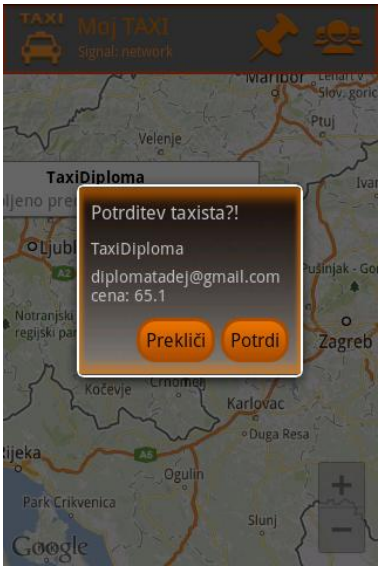


Slika 24: Potrditev taksista ob naročilu stranke.

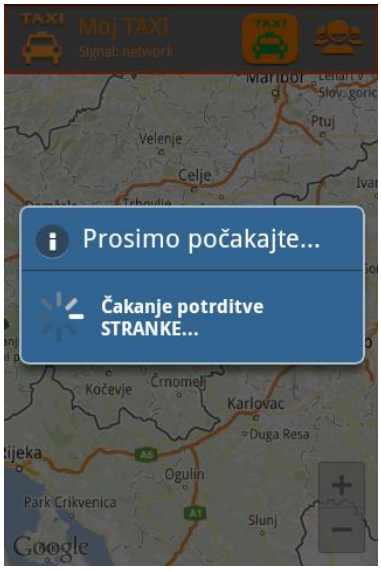


Slika 25: Čakanje stranke na potrditev naročila s strani taksista.

Ob uspešni komunikaciji med stranko in taksistom, aplikaciji na strani stranke zahteva še potrditev naročila. Stranka v potrditev pošlje taksistu sporočilo s sporočilom »STRANKA:OK«. V primeru, da si stranka premisli, lahko naročilo še vedno prekliče, s sporočilom »STRANKA:NE«.



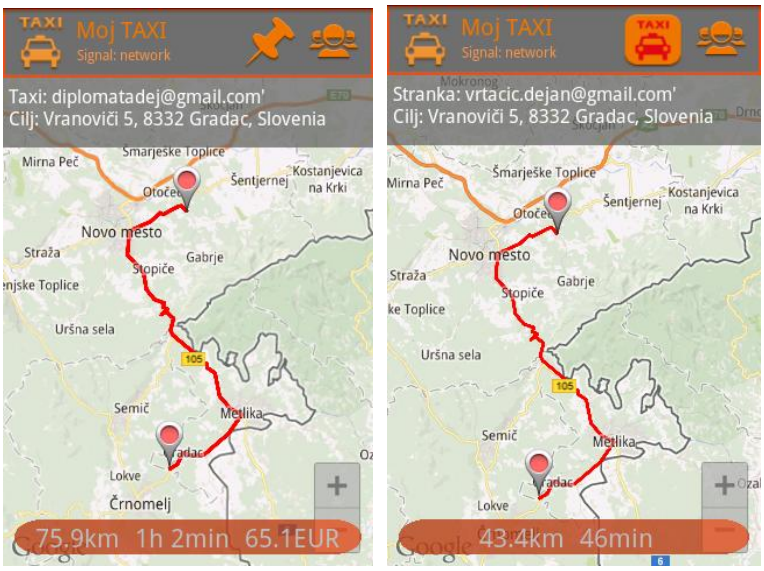
Slika 26: Dokončna potrditev ali preklic naročila stranke.



Slika 27: Čakanje taksista na potrditev naročila stranke.

V primeru potrditve naročila se taksistu na zemljevidu izriše pot do stranke ter pot do želenega cilja stranke.

Sami stranki pa se na zemljevidu izriše pot do svojega izbranega cilja. Prikazani si tudi podatki o prevozu. Podatki vsebujejo oddaljenost taksista od stranke (v kilometrih in v minutah), ceno prevoza ter ime izbranega taksista.



Slika 28: Izris poti na zemljevidu na strani stranke in taksista.

Posamezni segmenti sporočila so med seboj ločeni z dvopičjem, da jih lahko algoritem v aplikaciji pravilno prebere.

4.7 Nadzor taksistov preko spletne strani

V realnem življenju je v poslovnem svetu potreben nadzor nad delom delavcem. S tem si ustvarimo lahko neko analizo poteka dela zaposlenega in s pomočjo tega lahko optimiziramo oz. olajšamo način dela. Pomagamo tudi pri izboljšavi samega procesa dela in s tem tudi profita. Tudi v službi taksistov je tako. Vseskozi jih nadzira neka centralna postaja, ki jim nalaga delo in jih pravilno usmerja. Ker to delo naša aplikacija opravlja sama, smo si za bolj organizirano delo omislili statično spletno stran, ki prikazuje trenutne oz. nazadnje posodobljene lokacije vseh taksistov. S tem spremljamo razporeditev taksistov po nekem večjem mestu.

Spletna aplikacija je narejena s pomočjo naslednjih spletnih tehnologij:

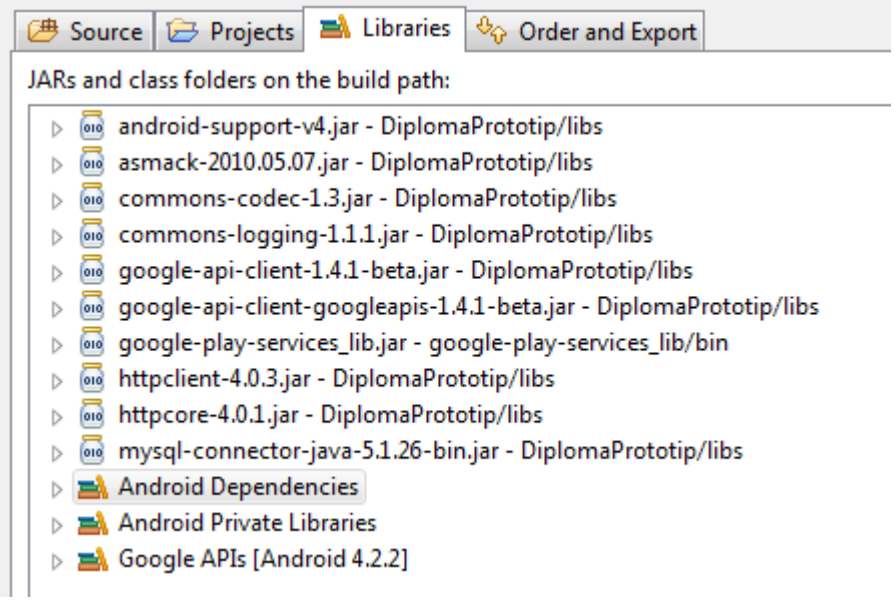
- *HTML (angl. HyperText Markup Language*) je glavni označevalni jezik za izdelavo spletnih strani in ostalih informacij, ki jih je moč prikazati na spletnem brskalniku.
- *CSS (angl. Cascading Style Sheets*), preprost slogovni jezik, ki skrbi za prezentacijo spletnih strani. HTML elementom definiramo njihov stil v smislu pravil, ki naj bi se prikazali na strani.
- *JavaScript* je objektni skriptni programski jezik, za pomoč pri ustvarjanju interaktivnih spletnih strani. Razvit je bil neodvisno od Jave, vendar si delita številne lastnosti in strukture.
- *PHP (angl. Hypertext Preprocessor*) izvira iz *Personal Home Page.* Je odprtokodni programski jezik za strežniške uporabe oziroma razvoj dinamičnih spletnih vsebin. Jezik deluje na več operacijskih sistemih in je zelo razširjen.

Spletna aplikacija omogoča prikaz lokacije taksistov in njihove ključne podatke. Osvežuje se samodejno z določenim časom, da imamo na voljo prikazane vedno najnovejše podatke taksistov. Smisel same aplikacije je oddaljen nadzor (centralna postaja). Nahaja se na spletnem naslovu: http://www.lovse.net/tadejvrtacic/nadzorTaksistov.html.

S spletno aplikacijo smo podali dobro idejo za nadaljnji, bolj kompleksen, razvoj mobilne aplikacije. Izboljšana spletna aplikacija bi spremljala kraje, kjer je potreba taksistov večja. Vsebovala bi algoritem, ki bi skrbel za optimizacijo lokacije taksista. Omogočili bi obveščanje predlagane lokacije posameznega taksista, ki je v sistemu z našo mobilno aplikacijo.

4.8 Težave pri razvoju prototipa

Pri razvoja prototipa so se vseskozi pojavljale napake. V začetku so se težave pojavile pri vključevanju knjižnic, ki smo jih potrebovali za razvoj. Nadgradnje in različne verzije operacijskega sistema Android hitro rastejo, s tem pa tudi verzije programskih knjižnic, ki jih potrebujemo pri razvoju. Tako smo jih morali medsebojno uskladiti in uporabiti pravo verzijo. V nasprotnem primeru so se pojavljale napake, pri katerih je bilo težko najti vir. Spodaj je prikazan nabor vseh knjižnic, uporabljenih v projektu, ki so med seboj kompatibilne.



Slika 29: Prikaz vseh knjižnic našega projekta aplikacije Android.

Ena večjih težav je nastopila že po razvoju delujočega prototipa. Namreč podjetje Google je svojo storitev Google Latitude, 9. Avgusta 2013, upokojilo. To je naš delujoč prototip aplikacije naredilo nedelujoč, kar je predstavljalo veliko težavo. V razvoj prototipa se je vložilo precej truda. Napoved podjetja Google, da bo v roku nekaj tednov upokojilo svojo storitev je bila za nas neprijetna. Jasno je bilo da bo potrebno storitev nadomestiti z novo alternativo, kar je nam vzelo še nekaj dodatnega časa. Storitev Google Latitude je v prototipu omogočala pridobivanje trenutnih lokacij taksistov ter zgodovino lokacij. Izdelali smo svojo alternativo, ki ima ekvivalentno delovanje, kot rešitev pred tem.

4.9 Nadgradnja in razširitve aplikacije

Praktično vsi telefoni z OS Android omogočajo hitro, enostavno posodobitev programske opreme, ki se iz meseca v mesec spreminja. Sicer dramatičnih sprememb ni, je pa opazna razlika hitrosti delovanja, porabi baterije itd.

Uporabniki Android so vse bolj zahtevnejši pri uporabi mobilne aplikacije. Mobilno aplikacijo je potrebno tudi iz tega razloga vseskozi dopolnjevati, nadgrajevati in optimizirati. Ta nadgraditev največkrat izhaja iz uporabniških izkušenj in se pokaže skozi čas uporabe aplikacije.

Preden si omislimo nadgradnjo, je potrebno izvesti analizo odzivnosti aplikacij. Največkrat se razvijalci sprašujemo: »Ali je naša poslovna aplikacija premalo odzivna?«. Oceniti treba izgubljeni čas uporabnikov naše poslovne aplikacije, zaradi njene preslabe odzivnosti. Zakasnitve v delovanju, v pomembni poslovni aplikaciji, povzročajo stroške, zato se je problema potrebno lotiti resno in odgovorno. Stroški spadajo med probleme, ki se odkrivajo

prepočasi. S tem se stanje lahko samo slabša. Pri nadgraditvi aplikacije je potrebno zbrati ustrezne informacije. Potrebna je optimalna prilagoditev mobilne aplikacije, za njeno učinkovito uporabo v poslovnem procesu.
Analiza odzivnosti aplikacije zagotavlja reševanje učinkovitosti aplikacije:

- Stalno ali občasno zmanjšanje odzivnosti aplikacije,
- nezaželene napake v zvezi z delovanjem aplikacije,
- uvedba novih tehnologij v informacijskem sistemu.

Odgovarja nam na ključna vprašanja, zakaj in kje se pojavljajo zakasnitve ali motnje aplikacije.

Za boljšo, hitrejšo in bolj funkcionalno aplikacijo bi poskrbeli z navedenimi izboljšavami:

- **Na strani koristnika oz. stranke taksistov:**
  - ✓ Aplikacija je zasnovana tako, da stranka s svojim naročanjem taksista avtomatsko, s pomočjo lokacijskih senzorjev, sporoči trenutno lokacijo. Tako taksist točno ve, kam mora po stranko, da jo popelje na želeni cilj. Ideja nadgradnje je, da ima stranka možnost že predhodno naročiti taksi. Npr. ob določeni uri, na določenem mestu. To bi omogočilo več dela taksistom in s tem večji zaslužek.
  - ✓ V osnovi aplikacija ne ponuja možnosti izbire cilja, ko je taksist že naročen. Nova funkcionalnost bi to omogočala. Namreč med čakanjem že naročenega taksista, bi si stranka lahko premislila in spremenila svojo končno lokacijo prevoza.
- **Na strani taksistov:**
  - ✓ Ko taksist potrdi zahtevo po prevozu stranke, se taksistu nastavi stanje o zasedenosti. S tem je taksist onemogočen za druge stranke. Z nadgradnjo bi to odpravili. Namreč v času prevoza stranke, se lahko pojavijo druge stranke, ki bi se vzporedno vključile v ta proces oz. prevoz več strank naenkrat. S tem bi zmanjšali tudi stroške strank, saj bi lahko za ceno ene stranke, omogočal to za dve ali več.
  - ✓ Lahko se zgodi, da naročen taksist, zaradi različnih dejavnikov, ne more opraviti storitve prevoza. V tem primeru bi imel vsak taksist možnost prevoz preložiti na svojega sodelavca, torej drugega taksista.
- **Na strani centrale (nadzor s spletno aplikacijo):**
  - ✓ Kje se naj taksist največ giblje v času iskanja novih strank, mu velevajo predvsem njegove izkušnje. Začetniki pa imajo pri tem lahko resen problem. Nadgradnja bi olajšala delo, tako začetnikom kot tudi »starim mačkom«. Izboljšana verzija bi vsebovala algoritem, ki bi na poljubnem strežniku neprestano preverjal lokacije dosegljivih taksistov in jim dajal navodila o njihovi najbolj optimalni razporeditvi. To razporeditev bi algoritem preverjal na podlagi izkušenj iz preteklosti. Algoritem bi omogočal, da smo ob pravem času na pravem mestu.

4.10 Testiranje aplikacije

Mobilno aplikacijo je potrebno testirati z namenom odkrivanja napak, ki so bile storjene pri načrtovanju ali izdelavi. Testiranje je eden od ključnih procesov, ki poda informacijo o kakovosti ter oceno, ali je aplikacija sprejemljiva za končne uporabnike. S testiranjem še vedno ne moremo zagotoviti popolnost in brezhibnost aplikacije, se pa s tem izboljšuje kakovost in zanesljivost mobilne aplikacije. Testiranje je zelo pomembno in ga je potrebno izvajati v zgodnji fazi razvoja. S tem lažje odkrivamo napake in stroški popravljanja napak so bistveno manjši. Testiranje je analiza produkta in primerjava med dejanskim stanjem produkta ter med pričakovanim oziroma zahtevanim. Natančno število napak je vnaprej nemogoče predvidevati. S testiranjem jih odkrijemo, da jih nato lahko odpravimo. Ko aplikacijo testiramo moramo predpostavljati da vsebuje napake. Vprašanje ki si ga razvijalci lahko vseskozi postavljamo je, kdaj je aplikacija dovolj testirana. Odgovor je, toliko časa kolikor bo potrebno za odkrivanje »vseh« napak. Če po nekaj sto urah delovanja programa ne odkrijemo niti ene napake je naša aplikacija dobro testirana. Lastnosti dobrega testa so: velika verjetnost odkritja napake, test ni odvečen, je najboljši test določenega tipa, ni preveč enostaven in ne preveč kompleksen.

Pri naši mobilni aplikaciji smo se najprej lotil testiranje enot oziramo posameznih segmentov. Pri razvijanju smo sprotno testirali posamezne module, metode, razrede ter aktivnosti. Pojavljale so se razne napake, ki smo jih s sprotnim testiranjem odpravili. Največ težav je nastopilo pri testiranju že razvitega prototipa. Testirali smo v realnem svetu z realnimi podatki in pogoji. Pri testiranju mobilne aplikacije je bilo potrebno obravnavati veliko dejavnikov. Najprej se pokaže zmogljivost naše mobilne naprave na kateri izvajamo test. Starejše mobilne naprave s starejšo verzijo OS Android in slabšimi karakteristikami, bi imele težave. V tem primeru bi bila mobilna aplikacija slabo odzivna, kar je za uporabnika zelo moteče. V nadaljevanju smo testirali signal GPS in signal mobilnega omrežja. To sta dve ključni stvari za delovanje naše mobilne aplikacije. Izguba signala GPS še ni pokazala večjih težav. Njegovo pridobivanje virov o trenutni lokaciji, je učinkovito nadomestilo omrežje. Večja težava je bila izguba omrežja. S tem je naša aplikacija postala nedelujoča in bi potrebovala nadgradnjo. Primer nadgradnje bi bilo shranjevanje podatkov na naš mobilni telefon, v času dobre povezave z omrežjem. Tako bi zmanjšali pridobivanje informacij oddaljenega strežnika. Z odpravo težave izgube signala omrežja, pa se pojavljajo nove težave. Ena izmed njih je realnost podatkov. Pri aplikaciji moramo zagotoviti da so podatki čim bolj točni in realni. Tolaži nas to, da bi v večjih mestih, kjer so taksisti najbolj uporabljeni, do izpada prišlo težje. V mestih, kot je Ljubljana, imamo poleg pokritosti mobilnega omrežja tudi dobro pokritost brezplačnega brezžičnega omrežja. To nam zagotavlja še hitrejše delovanje prenosa podatkov med strežnikom in aplikacijo. Komunikacija poteka bolj nemoteno, kar nudi zadovoljstvo uporabnikom.

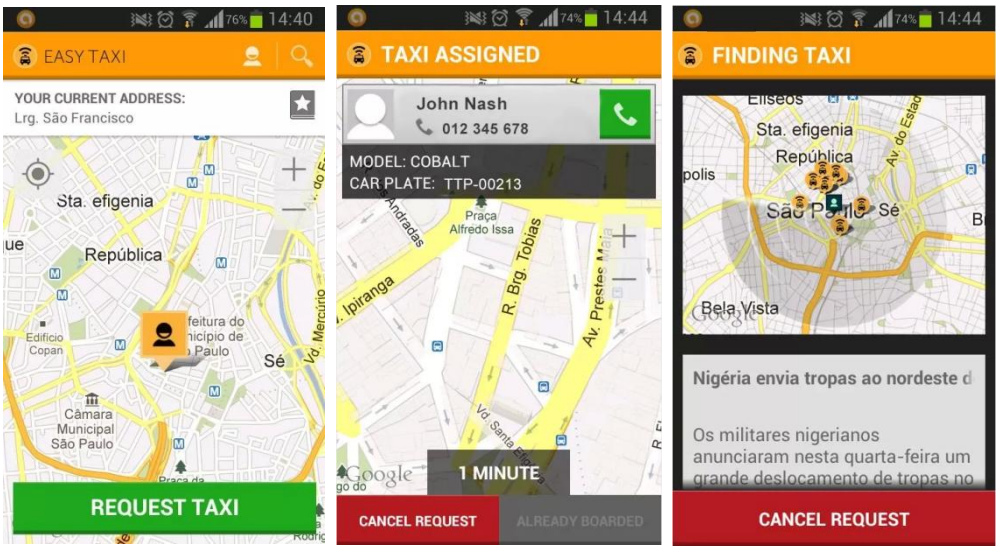
Testiranje z večimi mobilnimi napravami, z našim prototipom, je bilo sprva zanimivo. A se je kmalu pokazalo, da zna biti zelo zahtevno. Da aplikacijo naredimo dobro in je njeno delovanje kar se da optimalno, potrebujemo čas. Šele skozi čas nam delovanje aplikacije v realnosti poda želene informacije za izboljšavo ali odpravo napak.

Testiranje našega prototipa smo opravili z naslednjima mobilnima telefonoma:

- AVD (angl. Android Virtual Device)
  - Emulator Android verzije 2.3.3 z različnimi nastavitvami ločljivosti.
- Mobilni telefon Samsung Galaxy ACE S5830:
  - OS Android 2.3.3,
  - 320 x 480 točk, 16 milijonov barv, 3.5 palični zaslon,
  - procesor 800Hz in 278 MB pomnilnika (RAM),
  - 158MB internega pomnilnika.
- Mobilni telefon Samsung I9000 Galaxy S:
  - OS Android 2.3.4,
  - 480 x 800 točk, 16 milijonov barv, 4.0 palični zaslon,
  - procesor 1GHz in 512 MB pomnilnika (RAM),
  - 8GB internega pomnilnika.

5 OBSTOJEČE REŠITVE

Na slovenskem trgu še nisem zasledil mobilne aplikacije, ki bi omogočala podobne funkcionalnosti našega prototipa. Podporo taksistom preko mobilne aplikacije z OS Android na slovenskem trgu ponuja podjetje Taxi Laguna. Ponuja nam naročanje taksista z izpolnjenim obrazcem, kjer je potrebno vnesti želeni naslov, telefonsko številko, število oseb, način plačila ter datum in čas prevzema potnikov. Naročilo podjetje potrdi s potrditvenim SMS sporočilom. Številne mobilne aplikacije, za podporo taksistom v danem trenutku, pa obstajajo izven slovenskega trga. Na spletni strani aplikacij Android, imenovane Google Play, sem zasledil nekaj podobnih rešitev. Najbolj zanimivi sta aplikaciji, ki sta namenjeni zgolj uporabnikom oz. strankam taksistov določenih večjih mest. Prva aplikacija je Easy Taxi[10], ki je brezplačna in nam ponuja nabor dosegljivih taksistov v določenih mestih. Aplikacija je dosegljiva v Braziliji, Mexico, Columbia, Peru, Venezuela, Argentina, Chile, South Korea in Malaysia. Aplikacija avtomatsko zazna našo trenutno lokacijo in naš naslov. Nato lahko zahtevamo taksista s pritiskom na »request taxi«. Potrebno je izpolniti tudi referenčna polja, ki jih zahteva aplikacija. Taksist bo nato na svoji poti, ki jo lahko spremljamo ali pa ga lahko tudi pokličemo. Aplikacija je enostavna za uporabo in med najbolj priljubljenimi aplikacijami taksi storitev.



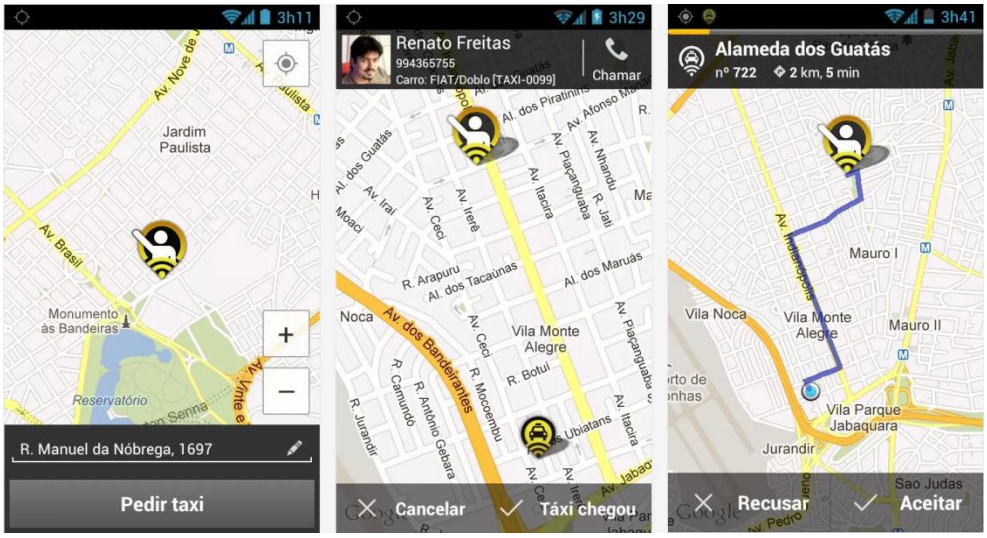
Slika 30: Primer brezplačne mobilne aplikacije Easy Taxi.

Mobilna aplikacija platforme Android, ki je vredna omembe in je namenjena zgolj taksistom je 99Taxis[11]. Na voljo je prav tako v Braziliji, in sicer v večjih mestih kot so Sao Paulo, Rio de janeiro, Belo Horizonte ... Njene prednosti so, da je aplikacija popolnoma brezplačna, in da sprejema plačilo storitve prevoza preko kreditnih kartic. Deluje na podoben princip kot naš prototip s strani taksista.

---

[10] https://play.google.com/store/apps/details?id=br.com.easytaxi&hl=sl,
[11] https://play.google.com/store/apps/details?id=com.nineninetaxis.driver&hl=sl

Aplikacija je namenjena zgolj taksistom. S pomočjo GPS pridobimo svojo trenutno lokacijo. Na zemljevidu se nam prikaže stranka, ki ima željo po prevozu. Taksistu se prikaže tudi pot do stranke, z njo pa lahko pridemo v stik tudi neposredno (direkten klic). Prikaz strank je prikazan na zemljevidu Google Maps. Komunikacija med taksistom in stranko poteka preko SMS sporočila. Aplikacija ima tudi drugo različico aplikacije 99Taxis, ki pa je namenjena zgolj strankam taksistov.



Slika 31: Primer brezplačne mobilne aplikacije 99Taxis.

6 SKLEP

Napredek v mobilni tehnologiji je očiten in se hitro razvija. Priljubljenost pametnih telefonov pa s tem narašča. S tem je prišla ideja o razvoju mobilne aplikacije na operacijskem sistemu Android. V sklopu diplomske naloge smo razvili prototip aplikacije Android za podporo taksistom. Uporabnik mobilne aplikacije bi storitev prevoza taksistov naročal kar preko pametnega telefona. Taksist pa bi preko mobilne aplikacije prejemal zahteve po naročilu storitve prevoza. Aplikacija omogoča lažjo ter boljšo komunikacijo taksistov z njihovimi strankami.

V diplomskem delu sem poskušal predstaviti brezplačne storitve Google ter njihovo avtentikacijo s protokolom OAuth 2.0. Njihov nabor je velik in kompleksen, ki ponuja veliko zadovoljstvo med uporabniki. V naši aplikaciji nam omogočajo hitro odzivnost in veliko funkcionalnost. Ključen potek aplikacije poteka na samem zemljevidu Google Maps. Zemljevid zagotavlja prikazovanje lokacij taksistov, izris poti, izbiro cilja stranke in še mnogo več. Za izmenjavo lokacij med uporabniki smo sprva uporabljali storitev Google Latitude. Nato smo zaradi njene upokojitve, morali težavo odpraviti s podobno rešitvijo. Nova rešitev je bila realizacija lastne podatkovne baze MySQL. Slednja omogoča shranjevanje in pridobivanje lokacij vseh taksistov. Sama komunikacija, med taksistom in koristnikom taksi službe, poteka preko storitve za komuniciranje, imenovano Google Talk.

Na slovenskem trgu je mobilnih aplikacij na tem področju malo. Zato je večja možnost, da bi se aplikacija razširila med uporabniki in bi ideja zaživela.

KAZALO SLIK

KAZALO TABEL

VIRI IN LITERATURA

[1] Peter Saint-Andre, Kevin Smith E, Remko Troncon, *XMPP: The Definitive Guide, O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol*, 2009, str. 11-22, 123, 124.

[2] Ryan Boyd, *Getting Started with OAuth 2.0, O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol,* 2012, str. 45-49 in 61.

[3] (2013) Operacijski sistem Android. Dostopno na: http://sl.wikipedia.org/wiki/Android_(operacijski_sistem).

[4] (2013) Android SDK Download. Dostopno na: http://developer.android.com/sdk/index.html.

[5] (2013) Android aplikacije. Dostopno na: http://android.fri.uni-lj.si/index.php/Android_Aplikacije.

[6] (2013) JDK (Java Development Kit). Dostopno na: http://en.wikipedia.org/wiki/Java_Development_Kit.

[7] (2013) Google APIs Client Library for Java. Dostopno na: https://developers.google.com/api-client-library/java/.

[8] (2013) RFC 5849: The Oauth 1.0 Protocol. Dostopno na: http://tools.ietf.org/html/rfc5849.

[9] (2013) The OAuht 1.0 Guide. Dostopno na: http://hueniverse.com/oauth/guide/.

[10] (2013) OpenID. Dostopo na: http://en.wikipedia.org/wiki/OpenID.

[11] (2013) RFC 6749: The OAuth 2.0 Authorization Framework. Dostopno na: http://tools.ietf.org/html/rfc6749

[12] (2013) Protokol Oauth 2.0. Dostopno na: https://code.google.com/p/google-oauth-java-client/wiki/OAuth2.

[13] (2013) Using OAuth 2.0 to Access Google APIs. Dostopno na: https://developers.google.com/accounts/docs/OAuth2?hl=sl.

[14] (2013) Using OAuth 2.0 for Installed Applications. Dostopno na: https://developers.google.com/accounts/docs/OAuth2InstalledApp.

[15] (2013) OAuth 2.0 Playground. Dostopno na: https://developers.google.com/oauthplayground/.

[16] (2013) OAuth 2.0 in Google Latitude. Dostopno na: http://blog.doityourselfandroid.com/2011/08/06/oauth-2-0-flow-android/.

[17] (2013) JSON (JavaScript Object Notaiton). Dostopno na: http://json.org/.

[18] (2013) Konzola Google APIs. Dostopno na: https://code.google.com/apis/console/?pli=1.

[19] (2013) Google APIs Console Help. Dostopno na: https://developers.google.com/console/help/#installed_applications.

[20] (2013) Google Drive SDK: Quickstart. Dostopno na: https://developers.google.com/drive/quickstart-android.

[21] (2013) XMPP: Instant messaging and Presence. Dostopno na: http://xml2rfc.tools.ietf.org/html/rfc6121.

[22] (2013) Protokol XMMP in primer povezave Google Talk. Dostopno na: http://developer.samsung.com/android/technical-docs/Building-a-Chat-Application.

[23] (2013) Google Maps. Dostopno na: http://en.wikipedia.org/wiki/Google_Maps.

[24] (2013) Google Maps Android API v2. Dostopno na: https://developers.google.com/maps/documentation/android/start.

[25] (2013) Google Latitude API. Dostopno na: https://developers.google.com/latitude/.

[26] (2013) GPS (Global Positioning System). Dostopno na: http://en.wikipedia.org/wiki/Global_Positioning_System.

[27] (2013) MySQL. Dostopno na: http://sl.wikipedia.org/wiki/MySQL.

[28] (2013) Android: Connecting to Server (MySQL using PHP). Dostopno na: http://m-zeeshanarif.blogspot.com/2013/05/android-connecting-to-server-mysql.html.

[29] (2013) Navigating the Social Terrain with Google Latitude. Dostopno na: http://nora.lis.uiuc.edu/images/iConferences/2010papers2_Page-Zhang.pdf.

[30] (2009) Google Latitude – Vdor v zasebnost? Dostopno na: http://www.mojmikro.si/v_srediscu/tehnologije/google_latitude-vdor_v_zasebnost.

[31] (2012) Mobile Application for News and Interactive Services. Dosotpno na: http://www.ejournalofscience.org/archive/vol2no1/vol2no1_1.pdf.